PrUcess (Processing unit through UART)

Introduction

Pr**U**cess is a **processing** unit that executes commands (arithmetic & logical operations, register file read & write operations) which are received from an external source through **UART** receiver module and it transmits the commands' results through the **UART** transmitter module.

This is a full ASIC design project (from RTL to GDS). It goes through the ASIC design flow from frontend to backend:

1. System's architecure design.
2. Synthesizable Verilog RTL modelling (behavioral modelling, structural modelling, and FSM coding) of all the system blocks from scratch (UART transmitter and receiver, integer clock divider, ALU, register file, parametrized data and bit synchronizers for solving CDC issues, reset synchronizer, and system's main controller).
3. Functional verification using self-checking testbenches and automated Python verification environments and running the testbenches using Modelsim.
4. Logic synthesis using Synopsys Design Compiler.
5. Formal verification post logic synthesis using Synopsys Formality.
6. Design for testatbility (DFT) using Synopsys DFT Compiler.
7. Formal verification post DFT using Synopsys Formality.
8. Physical design (floor planning, power planning, placement, CTS, routing, and chip finishing) using Cadence innovus.
9. Formal verification post physical design using Synopsys Formality.

Table of Contents

System's Specifications

UART is a standard serial communication protocol widely used in many applications. Oversampling is a technique used in UART receivers to improve the accuracy and reliability of the received data. In a UART receiver, data is received as a series of binary bits that are transmitted asynchronously with respect to a clock signal. To correctly interpret the received data, the receiver must sample the incoming signal at the correct time to capture the correct value of each bit. Oversampling involves sampling the incoming signal at a higher frequency than the baud rate of the transmitted data. This means that multiple samples are taken during the transmission of each bit, allowing the receiver to more accurately determine the timing and value of each bit. Oversampling also helps to mitigate the effects of noise and other signal distortions that can cause errors in the received data. By taking multiple samples of each bit, the receiver can detect and correct for these errors, improving the overall reliability of the data transmission.

The system includes two asynchronous clock domains (reference clock domain and UART clock domain). The command is received by the UART receiver, then it is sent to the system controller through a synchronizer (to solve the CDC issues) to decode and execute the command and then it sends the result to the UART transmitter through a synchronizer which will finally transmit it serially.

System's Parameters

Parameter	Default Value	Description
DATA_WIDTH	8	It is the size of: registers, ALU operands, UART transmitter frames, and UART receiver frames.
REGISTER_FILE_DEPTH	16	The number of registers in the register file.
SYNCHRONIZER_STAGE_COUNT	2	The number of stages in a synchronizer.
oversampling_prescale	the default value after resetting the system is 8	The ratio between the frequency of the UART receiver clock and the frequency of the UART transmitter clock.

System's Clock Domains

Clock Domain	Clock Names	Modules	Frequency
Reference clock domain	• reference_clk • ALU_clk	• System controller • ALU • Register file	Reference clock frequency = ALU clock frequency = 40 MHz

UART clock domain	• UART_clk • UART_transmitter_clk	• UART transmitter • UART receiver • Clock divider	• UART transmitter clock frequency = 115.2 KHz (standard baud rate) • UART clock frequency = oversampling_prescale * 115.2 KHz = 32 * 115.2 KHz = 3.6864 MHz

Note that the oversampling prescale can have the values (8, 16, or 32) but 32 is used in the simulations and backend flow to ensure that the UART receiver is functioning correctly in the worst case (highest clock frequency).

System's Components:

1. UART: It consists of UART receiver which receives the commands and UART transmitter that transmits the commands' results.
2. Clock divider: An integer clock divider which can divide the source clock up to division ratio of 32. It is used to divide the UART clock to produce UART transmitter clock with division ratio equal oversampling prescale.
3. ALU: It executes arithmetic and logical operations.
4. Clock gating cell: It is used to gate the ALU clock because there is significant time in which the ALU is not in operation (because the ALU operates on a very fast clock compared with the UART, so it waits long time to receive a new command).
5. Register file.
6. System controller: It is the main controller of the system. It consists of UART transmitter controller and UART receiver controller. The UART transmitter controller controls the UART transmitter by sending to it the data to be sent serially after it is ready (ALU result or register file data). The UART receiver controller controls the ALU and register file control signals based on the received frames from the UART receiver.
7. Reset synchronizer: It is used to synchronize the global reset to all clock domains.
8. Bus synchronizer: This module can be used to synchronize a single bit or a grey encoded bus between two asynchronous clock domains. It is a generic module (setting BUS_WIDTH = 1, means that it is a single bit synchronizer).
9. Data synchronizer: It is used to synchronize a bus by using a bit synchronizer and pulse generator to synchronize the bus's data valid signal.

ALU Operations:

1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Bit-wise AND (&)
6. Bit-wise OR (|)
7. Bit-wise NAND (~&)
8. Bit-wise NOR (~|)

9. Bit-wise XOR (^)
10. Bit-wise XNOR (~^)
11. Is equal (==)
12. Is greater than (>)
13. Is less than (<)< li>
14. Shift right (>>1)
15. Shift left (<<1)< li>

System's Commands:

- Register file write command. This command consists of 3 frames as follows:
    1. Command opcode (0xAA)
    2. Register file write address
    3. Register file write data
- Register file read command. This command consists of 2 frames as follows:
    1. Command opcode (0xBB)
    2. Register file read address
- ALU operation with operands command. The operands of the ALU are connected to the first two registers of the register file, so to execute this command: the operands are first written to the first two registers in the register file then the result is evaluated. This command consists of 4 frames as follows:
    1. Command opcode (0xCC)
    2. Operand A
    3. Operand B
    4. ALU function
- ALU operation without operands command. This command executes the ALU operation on the stored values in the first two registers in the register file directly. This command consists of 2 frames as follows:
    1. Command opcode (0xDD)
    2. ALU function

In all ALU commands, the UART transmitter sends two consecutive frames (becuase the size of the ALU result is double the size of the frame).

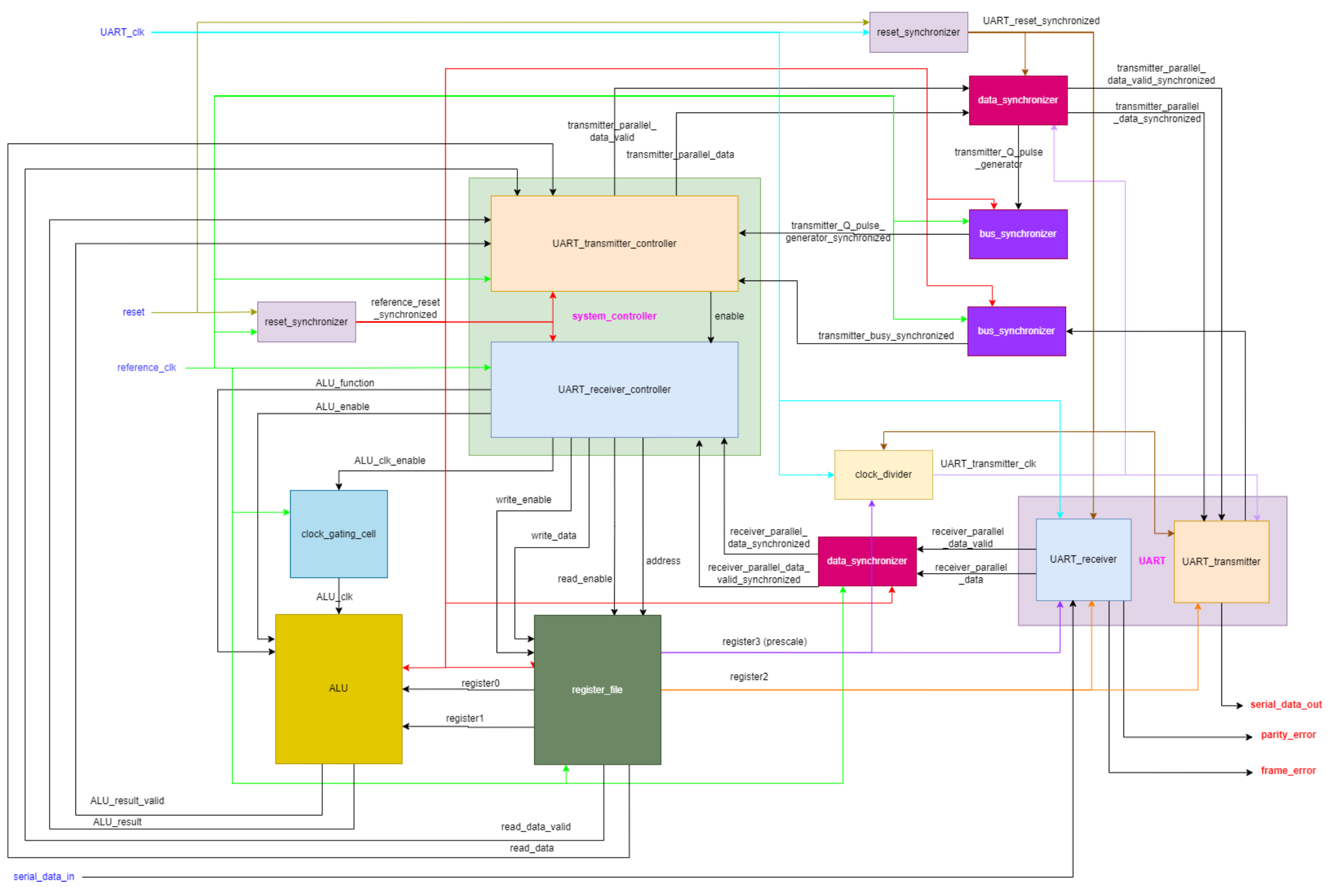
System's Configurations

- The parity configuration of UART (parity enable and parity type).
- The oversampling prescale (division ratio) of the UART receiver.

Note that the mentioned configurations are outputs from the register file (reference clock domain) and they are inputs to blocks that operates on UART clock (i.e. Metastability may occur becuase the source and destination domains are asynchronous to one another), however there is no synchronizers used to synchronize those signals because they are **Quasi-static signals** (they are effectively stable for long periods of time. Such domain crossings do not require synchronizers in the destination domain, because they are held long enough to be captured by even the slowest clock domains without the risk of metastability).

System Top Level Module

Block Diagram



Port Description

Port	Direction	Width	Description
reference_clk	input	1	The main clock of the system.
UART_clk	input	1	UART clock (the clock of the UART receiver).
reset	input	1	Unsynchronized global active low asynchronous reset.
serial_data_in	input	1	The data which is received serially by the UART receiver.
serial_data_out	output	1	The output of the UART transmitter (It is also the output of the mux that select between start, serial data, parity, or stop bits according to the state of the transmission).
parity_error	output	1	A signal to indicate that there is parity mismatch between the received parity bit and the calculated parity bit.
frame_error	output	1	A signal to indicate that the start bit or the stop bit was incorrect.

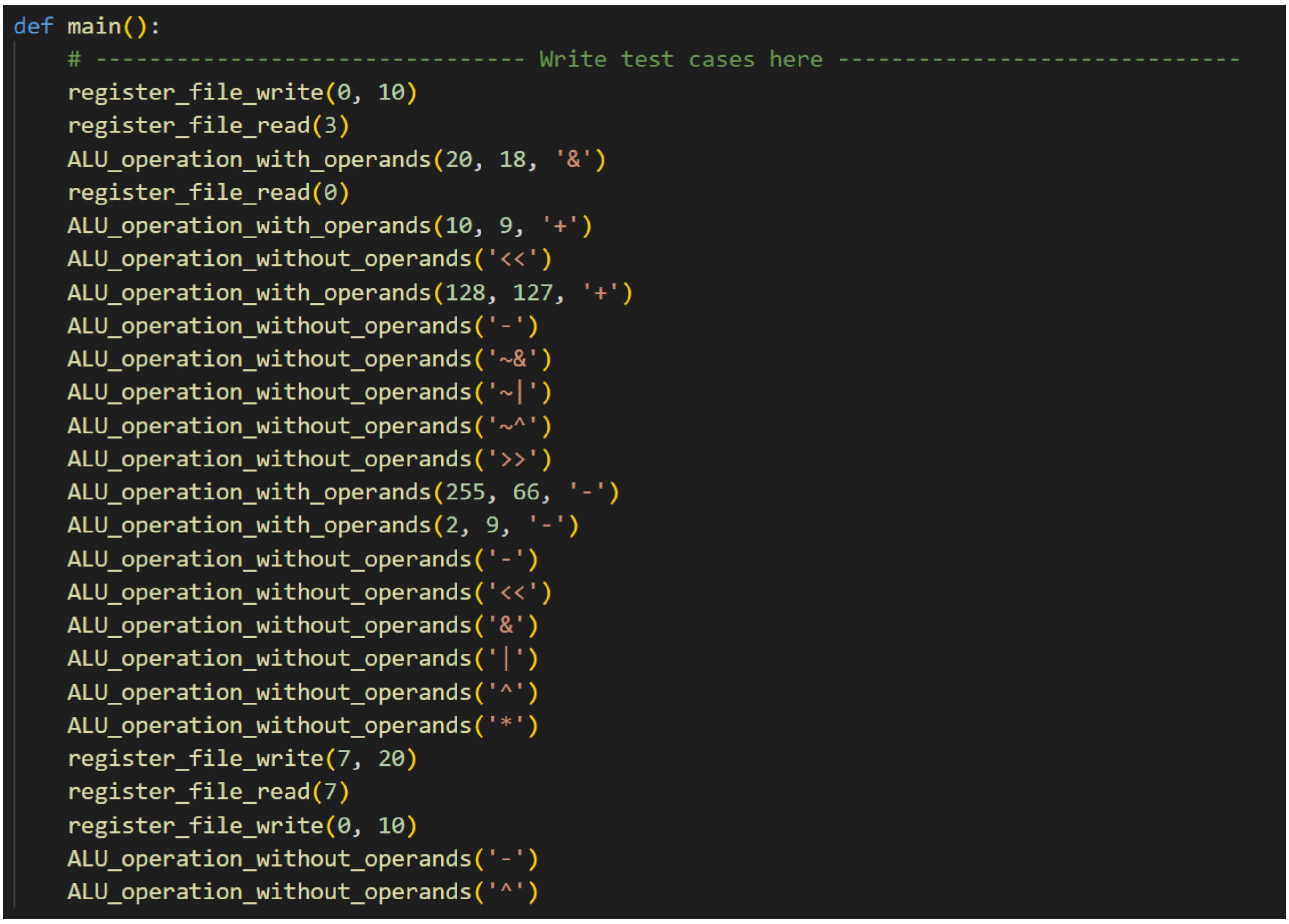
Functional Verification

The whole system is verified through an automated Python environment which does the following:

1. Generates the opcodes of all the given commands in an external file.
2. Generates all the expected results that should be transmitted serially through the UART transmitter in an external file.

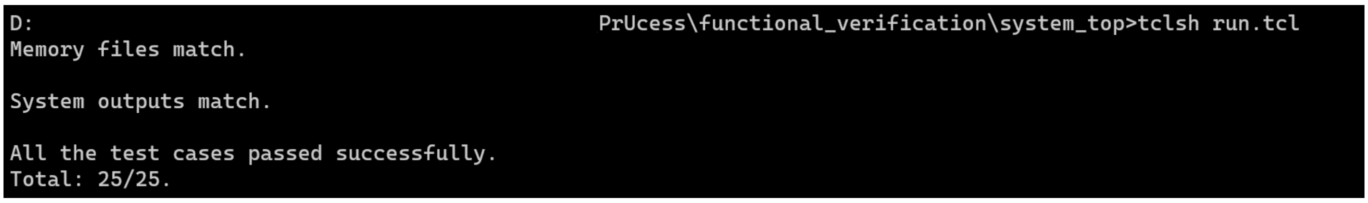
3. Generates the memory file which corresponds to the final values that should be stored in the register file after the execution of all the commands.
4. Compares the results of the Verilog testbench (transmitted through UART transmitter) and the generated memory file with the expected results' file and expected memory file.
5. Reports any mismatch that occur in the testbench.
6. Reports the number of passed and failed testcases.

Sample test cases: Any arbitrary test case can be written in `functional_verification/system_top/test_cases_generator.py` as the following:



```python
def main():
    # -------------------------------- Write test cases here ------------------------------
    register_file_write(0, 10)
    register_file_read(3)
    ALU_operation_with_operands(20, 18, '&')
    register_file_read(0)
    ALU_operation_with_operands(10, 9, '+')
    ALU_operation_without_operands('<<')
    ALU_operation_with_operands(128, 127, '+')
    ALU_operation_without_operands('-')
    ALU_operation_without_operands('~&')
    ALU_operation_without_operands('~|')
    ALU_operation_without_operands('~^')
    ALU_operation_without_operands('>>')
    ALU_operation_with_operands(255, 66, '-')
    ALU_operation_with_operands(2, 9, '-')
    ALU_operation_without_operands('-')
    ALU_operation_without_operands('<<')
    ALU_operation_without_operands('&')
    ALU_operation_without_operands('|')
    ALU_operation_without_operands('^')
    ALU_operation_without_operands('*')
    register_file_write(7, 20)
    register_file_read(7)
    register_file_write(0, 10)
    ALU_operation_without_operands('-')
    ALU_operation_without_operands('^')
```

Then the script `functional_verification/system_top/run.tcl` is used to: run the Python script to generate the expected results, run the testbench to generate the actual results, and compare both the results to report any mismatch in the results.



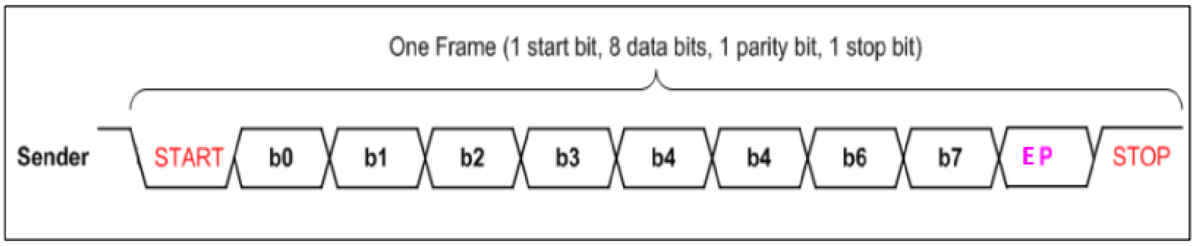
```
D:                                               PrUcess\functional_verification\system_top>tclsh run.tcl
Memory files match.

System outputs match.

All the test cases passed successfully.
Total: 25/25.
```

System Submodules

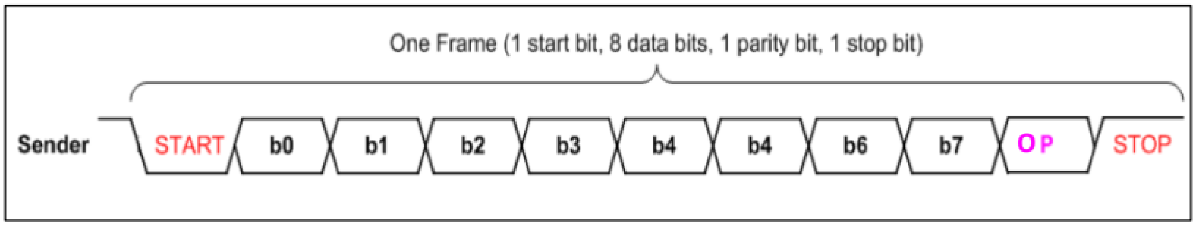
UART Transmitter

UART Transmitter Frame Types

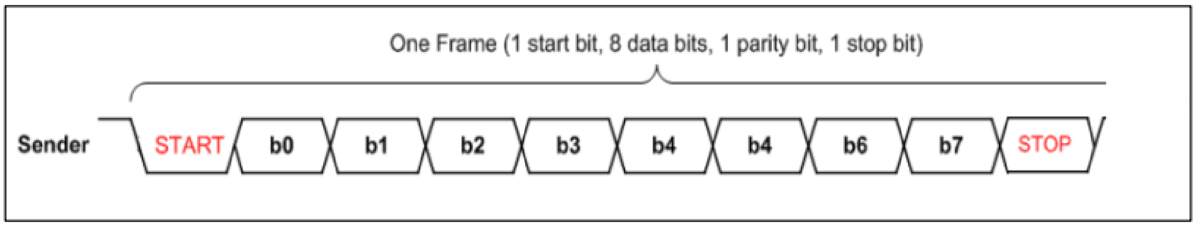
1. **Data Frame (in case of Parity is enabled & Parity Type is even)**
   - One start bit (1'b0)
   - Data (LSB first or MSB, 8 bits)
   - Even Parity bit
   - One stop bit


One Frame (1 start bit, 8 data bits, 1 parity bit, 1 stop bit)
Sender
START
b0
b1
b2
b3
b4
b4
b6
b7
EP
STOP

2. **Data Frame (in case of Parity is enabled & Parity Type is odd)**
   - One start bit (1'b0)
   - Data (LSB first or MSB, 8 bits)
   - Odd Parity bit
   - One stop bit

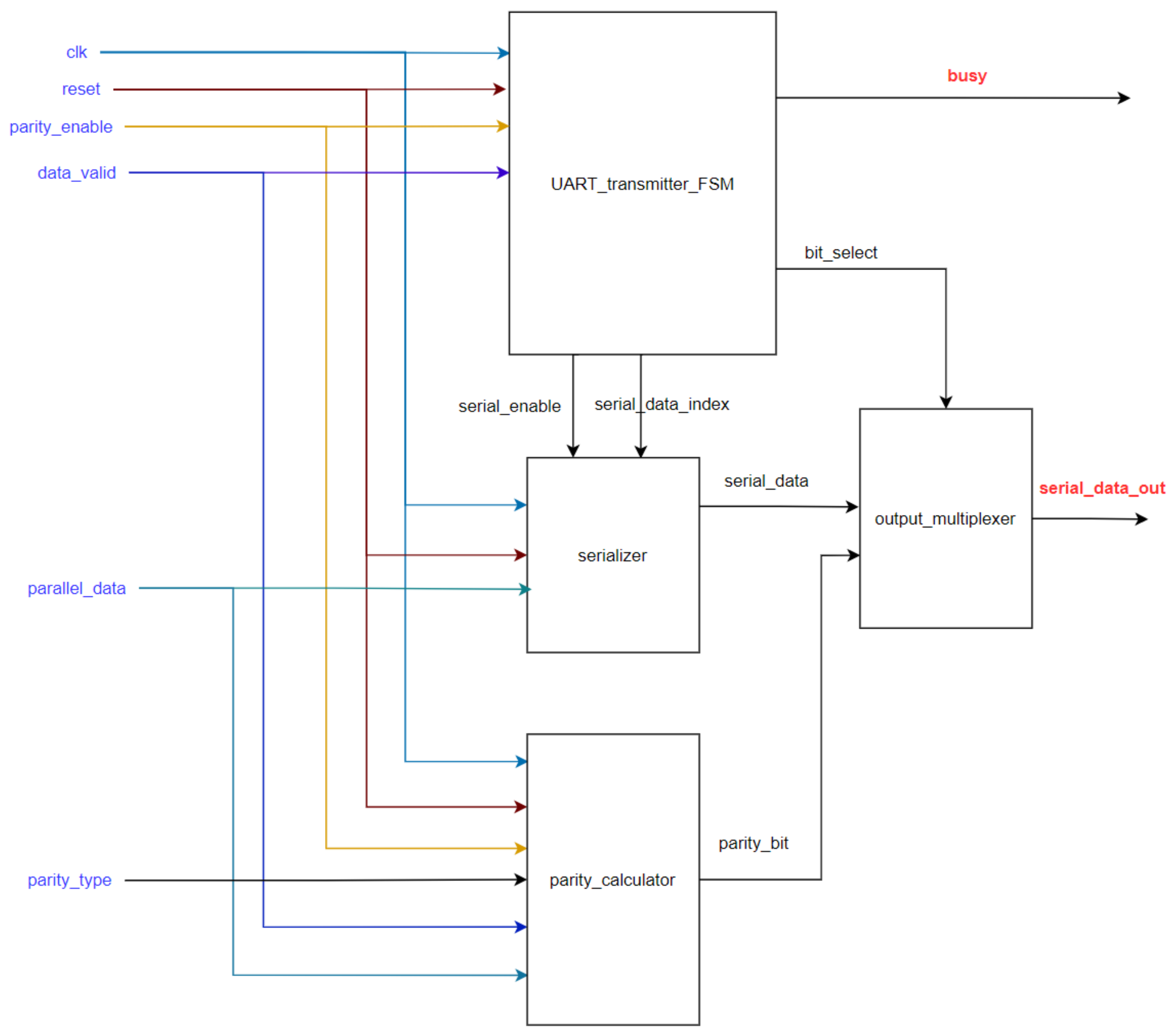

One Frame (1 start bit, 8 data bits, 1 parity bit, 1 stop bit)
Sender
START
b0
b1
b2
b3
b4
b4
b6
b7
OP
STOP

3. **Data Frame (in case of Parity is not Enabled)**
   - One start bit (1'b0)
   - Data (LSB first or MSB, 8 bits)
   - One stop bit


One Frame (1 start bit, 8 data bits, 1 parity bit, 1 stop bit)
Sender
START
b0
b1
b2
b3
b4
b4
b6
b7
STOP

Top Level Module

Block Diagram



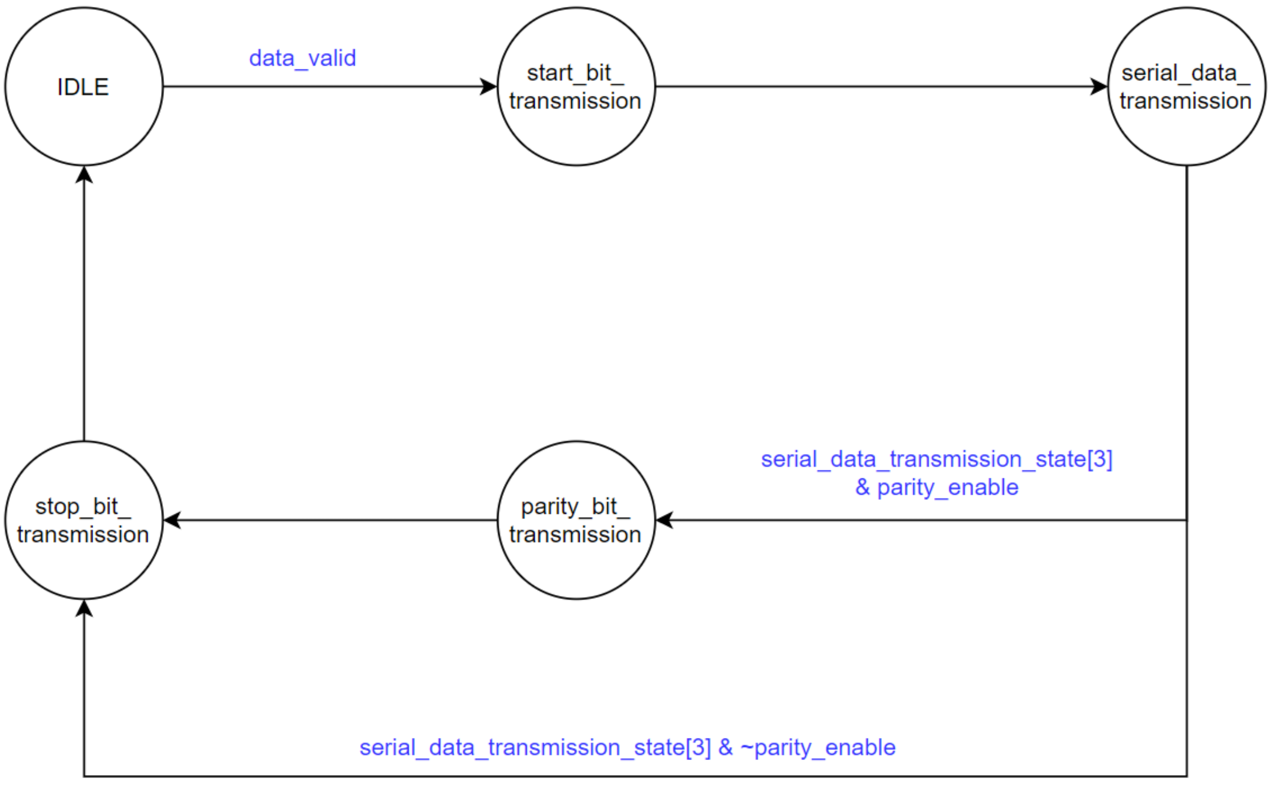
Port Description

Port	Direction	Width	Description
clk	input	1	Generated clock produced from the clock divider whose source clock is UART clock.
reset	input	1	Global active low asynchronous reset after synchronization.
parity_type	input	1	A signal to indicate the parity type (1 for odd, 0 for even).
parity_enable	input	1	A signal to enable the transmission of the parity bit in the frame.
data_valid	input	1	A signal that indicates that there exist data to be transmitted.
parallel_data	input	DATA_WIDTH (default value is 8)	The data to be transmitted by the UART transmitter.

serial_data_out	output	1	The output of the transmitter (It is also the output of the mux that selects between start, serial data, parity, or stop bits according to the state of the transmission).
busy	output	1	A signal that indicates that the transmitter is currently in operation and it can't transmit new data.

UART Transmitter Finite State Machine (FSM)

FSM


IDLE
data_valid
start_bit_ transmission
serial_data_ transmission
serial_data_transmission_state[3] & parity_enable
parity_bit_ transmission
stop_bit_ transmission
serial_data_transmission_state[3] & ~parity_enable

Note that if any omitted condition occurs, the current state won't change.

This FSM controls the following output ports according to the current state: busy, serial_enable, bit_select.

Port Description

Port	Direction	Width	Description
clk	input	1	Generated clock produced from the clock divider whose source clock is UART clock.
reset	input	1	Global active low asynchronous reset after synchronization.
parity_enable	input	1	A signal to enable the transmission of the parity bit in the frame.

data_valid	input	1	A signal that indicates that there exist data to be transmitted.
serial_enable	output	1	A signal to enable the operation of the serializer.
bit_select	output	2	The output mux selection bits that selects the output bit (start bit, serial data bit, parity bit, or stop bit). The value of those selection bits is decided according to the current state of the transmission.
seial_data_index	output	$\log_2$(DATA_WIDTH) (default value is 3)	A number between 0 and 7 that indicates the index of the bit to be transmitted serially.
busy	output	1	A signal that indicates that the transmitter is currently in operation and it can't transmit new data.

Serializer

It produces serial bits from the parallel data (from LSB to MSB) according to the input index.

Port Description

Port	Direction	Width	Description
clk	input	1	Generated clock produced from the clock divider whose source clock is UART clock.
reset	input	1	Global active low asynchronous reset after synchronization.
parallel_data	input	DATA_WIDTH (default value is 8)	The data to be transmitted by the UART transmitter.
serial_enable	input	1	A signal to enable the operation of the serializer.
serial_data_index	input	$\log_2$(DATA_WIDTH) (default value is 3)	A number between 0 and 7 that indicates the index of the bit to be transmitted serially.
serial_data	output	1	The bit that is serially transmitted from the UART transmitter.

Parity Calculator

It calculates the parity bit according to the parity type.

Port Description

Port	Direction	Width	Description
clk	input	1	Generated clock produced from the clock divider whose source clock is UART clock.

reset	input	1	Global active low asynchronous reset after synchronization.
parity_type	input	1	A signal to indicate the parity type (1 for odd, 0 for even).
parity_enable	input	1	A signal to enable the transmission of the parity bit in the frame.
parallel_data	input	DATA_WIDTH (default value is 8)	The data to be transmitted by the UART transmitter.
parity_bit	output	1	The parity bit of the parallel data to be transmitted.

Output Multiplexer

It selects between (start bit, serial data bit, parity bit, or stop bit) according to the current state of trasnmission.

Port Description

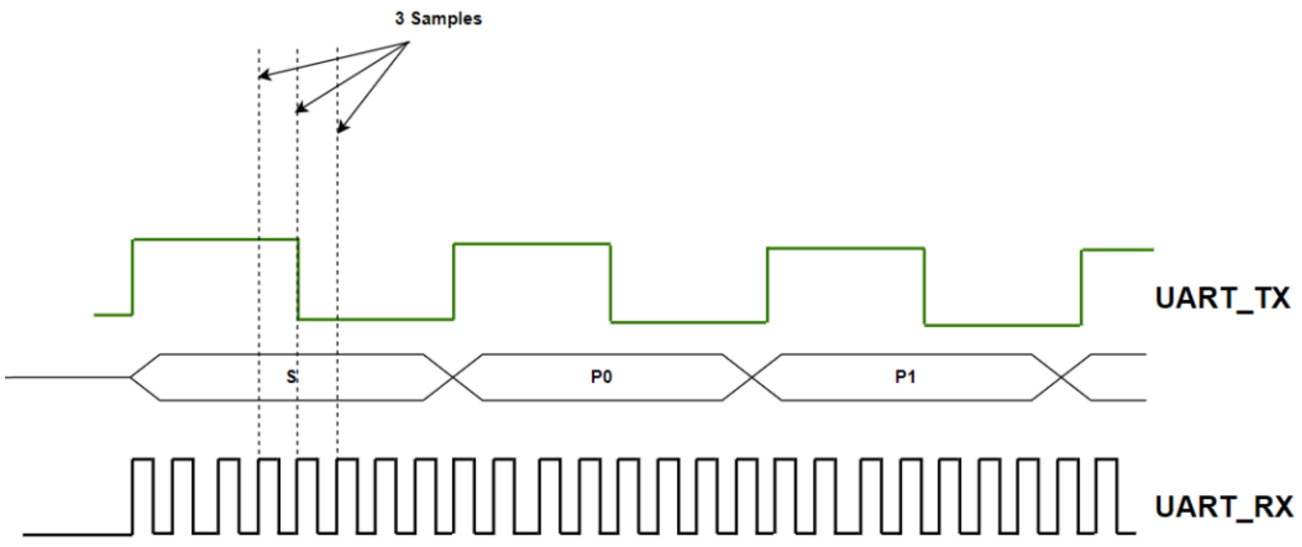
Port	Direction	Width	Description
bit_select	input	2	The output mux selection bits that selects the output bit (start bit, serial data bit, parity bit, or stop bit).
serial_data	input	1	The bit that is serially transmitted from the UART transmitter (i.e. the output bit from the serializer).
parity_bit	input	1	The parity bit of the parallel data to be transmitted.
mux_out	output	1	The output of the mux that selects between start, serial data, parity, or stop bits according to the state of the transmission.

Functional Verification

This module is verified through self-checking testbench in Modelsim. The testbench can be run using `run.tcl` script.

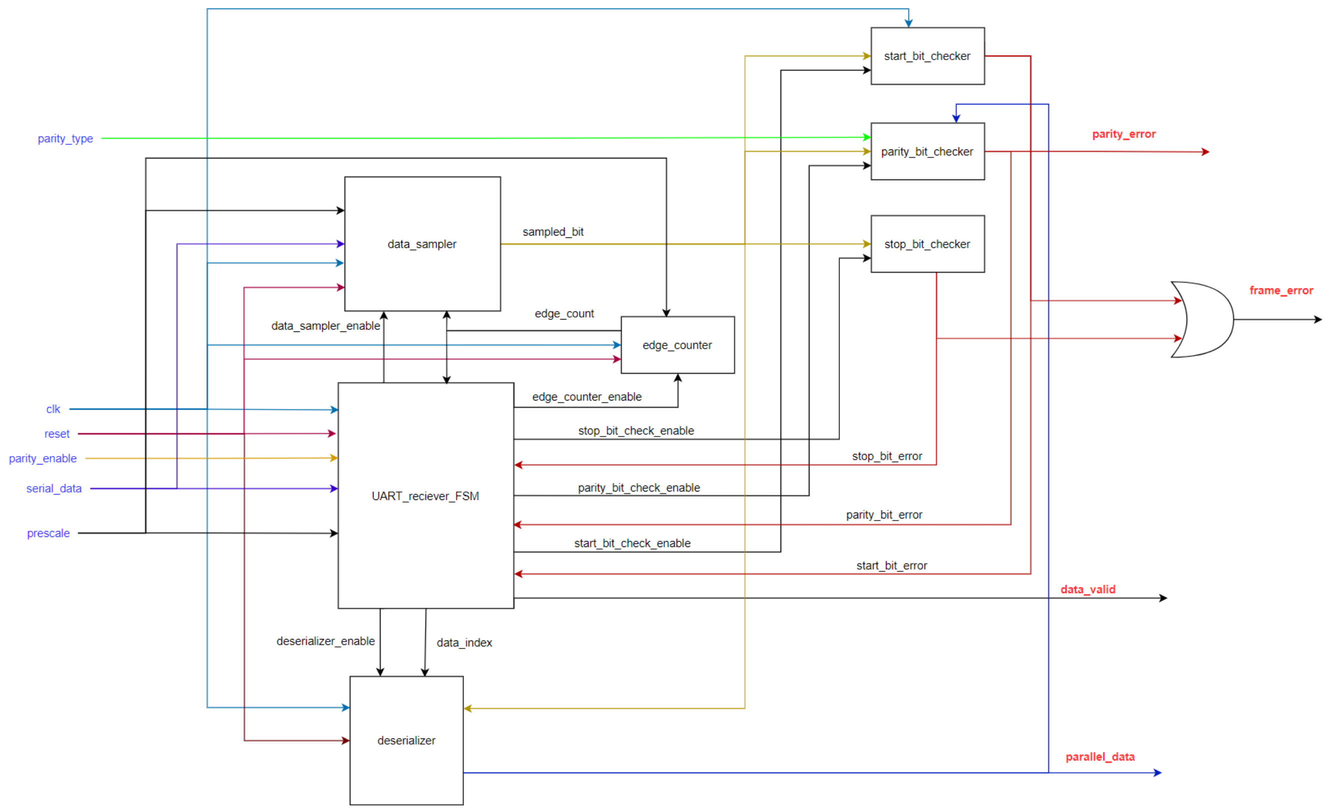
UART Receiver

The following figure illustrates how oversampling with prescale = 8 works, the recieved bit is sampled 3 times and output bit from the sampler is the most represented bit in those 3 bits.



Top Level Module

Block Diagram

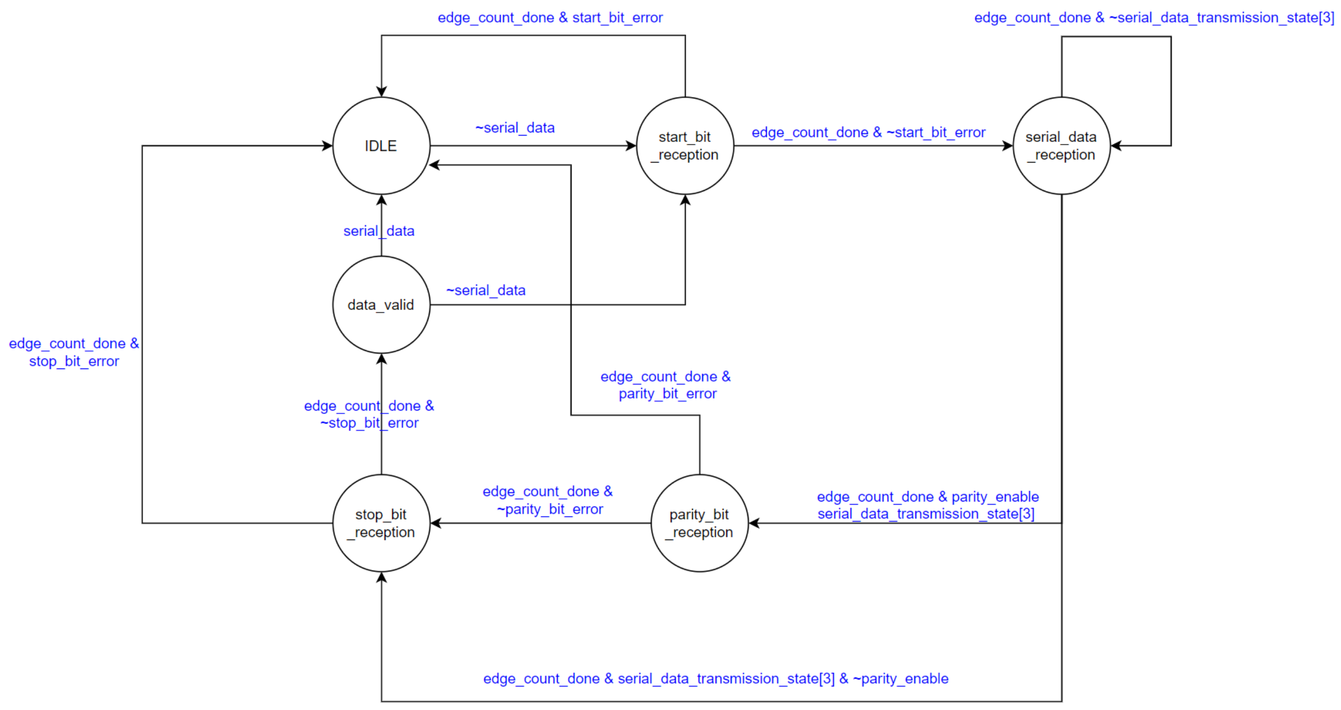


Port Description

Port	Direction	Width	Description
clk	input	1	UART clock.
reset	input	1	Global active low asynchronous reset after synchronization.

parity_type	input	1	A signal to indicate the parity type (1 for odd, 0 for even).
parity_enable	input	1	A signal to enable the transmission of the parity bit in the frame.
prescale	input	6	The ratio between the frequency of the receiver and the frequecy of the transmitter (The avaialable oversampling_prescale values are: 8, 16, 32).
serial_data_in	input	1	The data which is received serially.
data_valid	output	1	A signal to indicate that the received data was free of errors.
parallel_data	output	DATA_WIDTH (default value is 8)	The data which is received serially bit by bit.
parity_error	output	1	A signal to indicate that there is parity mismatch between the received parity bit and the calculated parity bit.
frame_error	output	1	A signal to indicate that the start bit or the stop bit was incorrect.

UART Receiver Finite State Machine (FSM)


edge_count_done & start_bit_error
edge_count_done & ~serial_data_transmission_state[3]
IDLE
~serial_data
start_bit _reception
edge_count_done & ~start_bit_error
serial_data _reception
serial_data
data_valid
~serial_data
edge_count_done & stop_bit_error
edge_count_done & parity_bit_error
edge_count_done & ~stop_bit_error
edge_count_done & ~parity_bit_error
edge_count_done & parity_enable serial_data_transmission_state[3]
stop_bit _reception
parity_bit _reception
edge_count_done & serial_data_transmission_state[3] & ~parity_enable

Note that if any omitted condition occurs, the current state won't change.

Port	Direction	Width	Description
clk	input	1	UART clock.
reset	input	1	Global active low asynchronous reset after

			synchronization.
parity_enable	input	1	A signal to enable the transmission of the parity bit in the frame.
prescale	input	6	The ratio between the frequency of the receiver and the frequecy of the transmitter (The avaialable prescale values are: 8, 16, 32).
serial_data_in	input	5	The data which is received serially.
start_bit_error	input	1	A signal to indicate that the sampled start bit is wrong (i.e. the samples are 011 or 111 or 110 or 101).
parity_bit_error	input	1	A signal to indicate that the sampled parity bit is wrong.
stop_bit_error	input	1	A signal to indicate that the sampled stop bit is wrong (i.e. the samples are 100 or 000 or 001 or 010).
edge_count	input	5	A counter value which indicates the number of the current edge. Its value depends on the prescale value (because prescale of value 8 means that the counter should stop at 7 and wrap around again).
edge_count_done	input	1	A signal to indicate that a full cycle of the UART tranmsitter has passed (when prescale value is 8, edge_count_done becomes high when the edge counter value is 7).
start_bit_check_enable	output	1	A signal to enable the operation of the start bit checker.
parity_bit_check_enable	output	1	A signal to enable the

			operation of the parity bit checker.
stop_bit_check_enable	output	1	A signal to enable the operation of the stop bit checker.
edge_counter_and_data_sampler_enable	output	1	A signal to enable the operation of the edge counter and data sampler.
deserializer_enable	output	1	A signal to enable the operation of the deserializer.
data_index	output	$\log_2$(DATA_WIDTH) (default value is 3)	The index of the of bit to be received in the frame.
data_valid	output	1	A signal to indicate that the received data by the UART receiver was free of errors.

Edge Counter

Port	Direction	Width	Description
clk	input	1	UART clock.
reset	input	1	Global active low asynchronous reset after synchronization.
prescale	input	6	The ratio between the frequency of the receiver and the frequecy of the transmitter (The avaialable prescale values are: 8, 16, 32).
enable	input	1	A signal to enable the operation of the edge counter.
edge_count	output	5	A counter value which indicates the number of the current edge. Its value depends on the prescale value (because prescale of value 8 means that the counter should stop at 7 and wrap around again).
edge_count_done	output	1	A signal to indicate that a full cycle of the UART tranmsitter has passed (when prescale value is 8, edge_count_done becomes high when the edge counter value is 7).

Data Sampler

Port	Direction	Width	Description
clk	input	1	UART clock.
reset	input	1	Global active low asynchronous reset after synchronization.

serial_data_in	input	1	The data which is received serially.
prescale	input	5	The ratio between the frequency of the receiver and the freqecy of the transmitter (The avaialable prescale values are: 8, 16, 32). These are the 5 MSBs of the prescale, becuase the data sampler module operates on the prescale after shifting its value.
enable	input	1	A signal to enable the data sampler.
edge_count	output	5	A counter value which indicates the number of the current edge. Its value depends on the prescale value (because prescale of value 8 means that the counter should stop at 7 and wrap around again).
sampled_bit	output	1	The resulting sampled bit out of three samples taken at three different edges. It is equal to the bit appearing the most times in the samples (e.g. if samples = 101, sampled_bit = 1. if samples = 100, sampled_bit = 0).

Deserializer

Port	Direction	Width	Description
clk	input	1	UART clock.
reset	input	1	Global active low asynchronous reset after synchronization.
enable	input	1	A signal to enable the operation of the deserializer.
data_index	input	$\log_2$(DATA_WIDTH) (default value is 3)	The index of the of bit to be received in the frame.
sampled_bit	input	1	The resulting sampled bit out of three samples taken at three different edges. It is equal to the bit appearing the most times in the samples (e.g. if samples = 101, sampled_bit = 1. if samples = 100, sampled_bit = 0).
parallel_data	output	DATA_WIDTH (default value is 8)	The data which is received serially bit by bit.

Parity Bit Checker

Port	Direction	Width	Description
clk	input	1	UART clock.
reset	input	1	Global active low asynchronous reset after synchronization.
parity_type	input	1	A signal to indicate the parity type (1 for odd, 0 for even).
enable	input	1	A signal to enable the operation of the parity bit checker.

sampled_bit	input	1	The sampled bit from the data sampler. It is always the parity bit for this module because it is only enabled when the parity bit is received.
parallel_data	input	DATA_WIDTH (default value is 8)	The data which is received serially bit by bit.
parity_bit_error	output	1	A signal to indicate that there is parity mismatch between the received parity bit and the calculated parity bit.

Start Bit Checker

Port	Direction	Width	Description
clk	input	1	UART clock.
reset	input	1	Global active low asynchronous reset after synchronization.
enable	input	1	A signal to enable the operation of the start bit checker.
sampled_bit	input	1	The sampled bit from the data sampler. It is always the start bit for this module because it is only enabled when the start bit is received.
start_bit_error	output	1	A signal to indicate that the start bit is incorrect (the sampled bit is 1).

Stop Bit Checker

Port	Direction	Width	Description
clk	input	1	UART clock.
reset	input	1	Global active low asynchronous reset after synchronization.
enable	input	1	A signal to enable the operation of the stop bit checker.
sampled_bit	input	1	The sampled bit from the data sampler. It is always the stop bit for this module because it is only enabled when the stop bit is received.
stop_bit_error	output	1	A signal to indicate that the stop bit is incorrect (the sampled bit is 0).

Functional Verification

This module is verified through self-checking testbench in Modelsim. The testbench can be run using `run.tcl` script.

Clock Divider

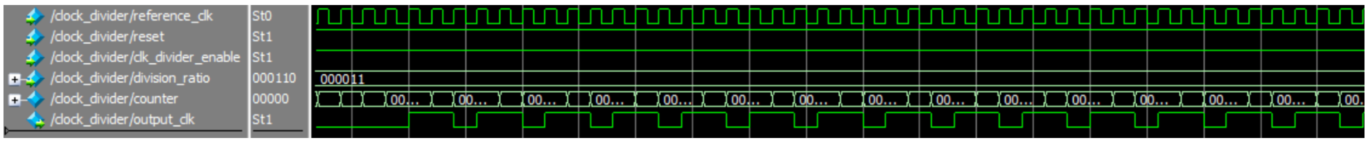
Port Description

Port	Direction	Width	Description
reference_clk	input	1	The source clock (UART clock).
reset	input	1	Global active low asynchronous reset after synchronization.
clk_divider_enable	input	1	An enable signal for the clock divider.
division_ratio	input	6	The division ratio of the clock divider (oversampling_prescale), it is connected to register_file[3].
output_clk	output	1	The output divided clock (UART_transmitter_clk).

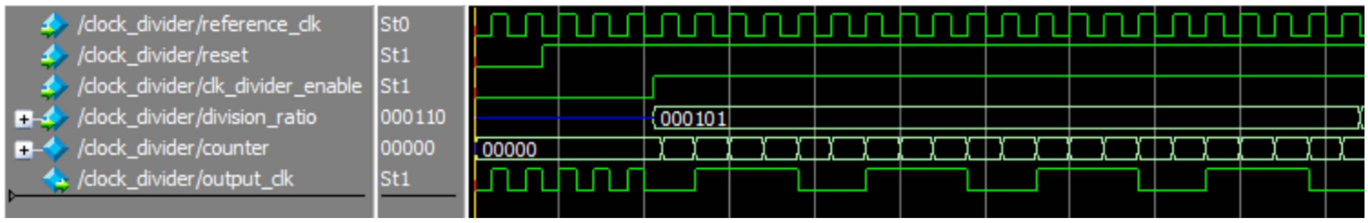
Functional Verification

This module is verified through logic simulation in Modelsim. The simulation can be run using `run.tcl` script.

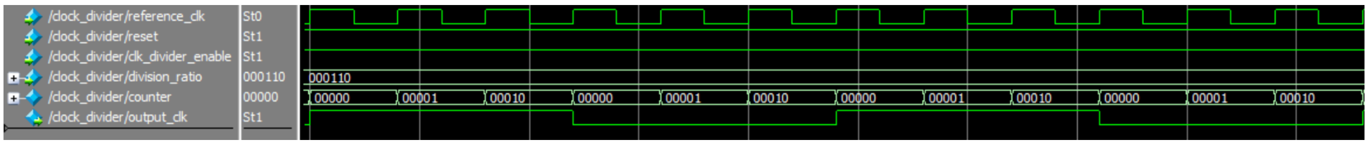
Test case (1) (division_ratio = 3)


/clock_divider/reference_clk St0
/clock_divider/reset St1
/clock_divider/clk_divider_enable St1
/clock_divider/division_ratio 000110
/clock_divider/counter 00000
/clock_divider/output_clk St1
000011

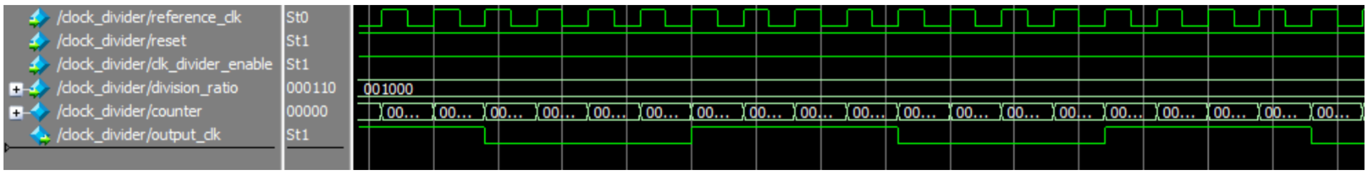
Test case (2) (division_ratio = 5)


/clock_divider/reference_clk St0
/clock_divider/reset St1
/clock_divider/clk_divider_enable St1
/clock_divider/division_ratio 000110
/clock_divider/counter 00000
/clock_divider/output_clk St1
000101
00000

Test case (3) (division_ratio = 6)


/clock_divider/reference_clk St0
/clock_divider/reset St1
/clock_divider/clk_divider_enable St1
/clock_divider/division_ratio 000110
/clock_divider/counter 00000
/clock_divider/output_clk St1
000110
00000
00001
00010
00000
00001
00010
00000
00001
00010
00000
00001
00010

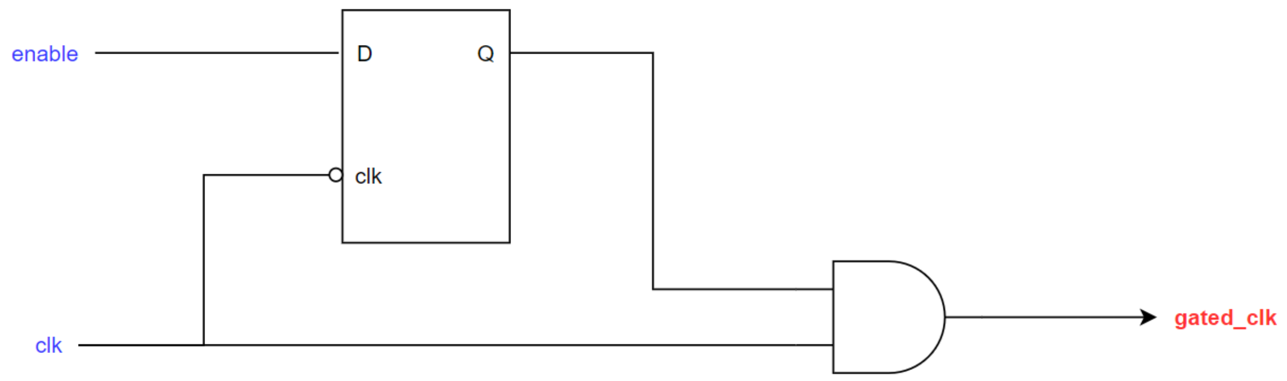
Test case (4) (division_ratio = 8)


/clock_divider/reference_clk St0
/clock_divider/reset St1
/clock_divider/clk_divider_enable St1
/clock_divider/division_ratio 000110
/clock_divider/counter 00000
/clock_divider/output_clk St1
001000

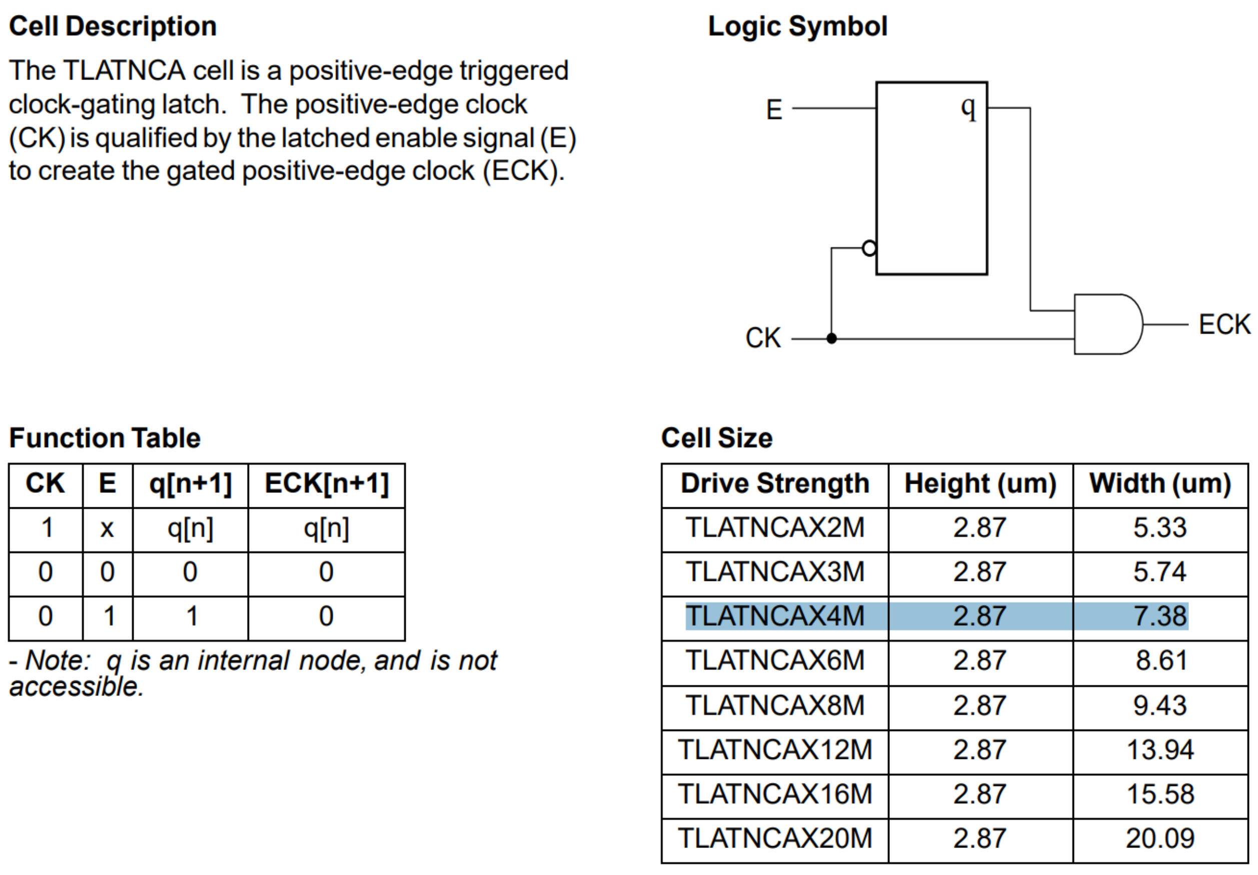
Clock Gating Cell

Logic Diagram

Latch-based clock gating cell:



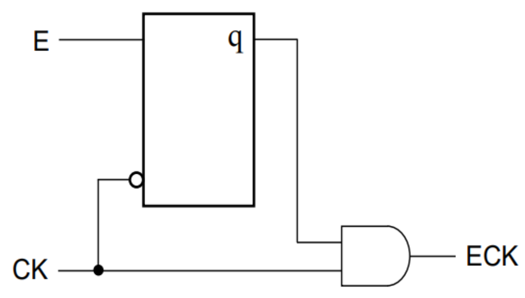
This module is replaced by the integrated clock gating cell (ICG) from the standard cell library. ICG is a full custom cell whose path delays are well balanced which eliminates the occurrence of pulse clipping and spurious clocking issues (those issues may occur only after fabrication due to imbalanced delays but not in simulation because ideally, the latch-based clock gating cell doesn't suffer from any clock issues). The replacement procedure is done automatically using the `place_ICG_cell.tcl` script to ease the process of placement (in the backend flow) and removal (in the functional simulation and verification) of the ICG cell.



Cell Description

The TLATNCA cell is a positive-edge triggered clock-gating latch. The positive-edge clock (CK) is qualified by the latched enable signal (E) to create the gated positive-edge clock (ECK).

Logic Symbol



Function Table

CK	E	q[n+1]	ECK[n+1]
1	x	q[n]	q[n]
0	0	0	0
0	1	1	0

- Note: q is an internal node, and is not accessible.

Cell Size

Drive Strength	Height (um)	Width (um)
TLATNCAX2M	2.87	5.33
TLATNCAX3M	2.87	5.74
TLATNCAX4M	2.87	7.38
TLATNCAX6M	2.87	8.61
TLATNCAX8M	2.87	9.43
TLATNCAX12M	2.87	13.94
TLATNCAX16M	2.87	15.58
TLATNCAX20M	2.87	20.09

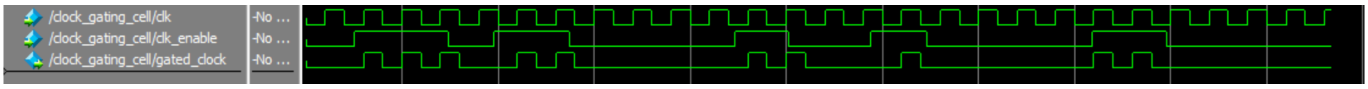
Port Description

Port	Direction	Width	Description
clk	input	1	The source clock to be gated (reference clock).

clk_enable	input	1	An enable signal for the clock gating.
gated_clk	output	1	The output gated clock (ALU clock).

Functional Verification

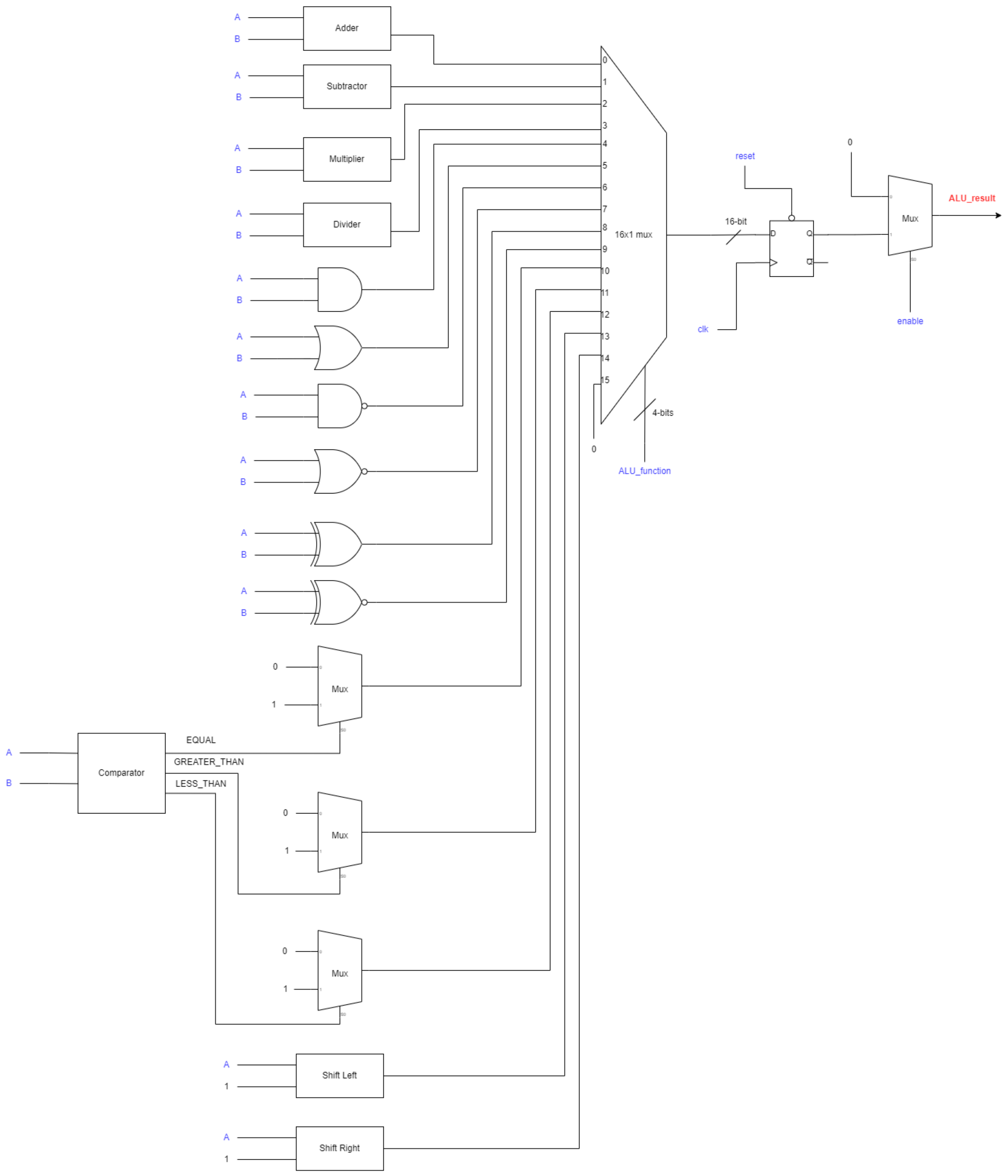
This module is verified through logic simulation in Modelsim. The simulation can be run using `run.tcl` script.



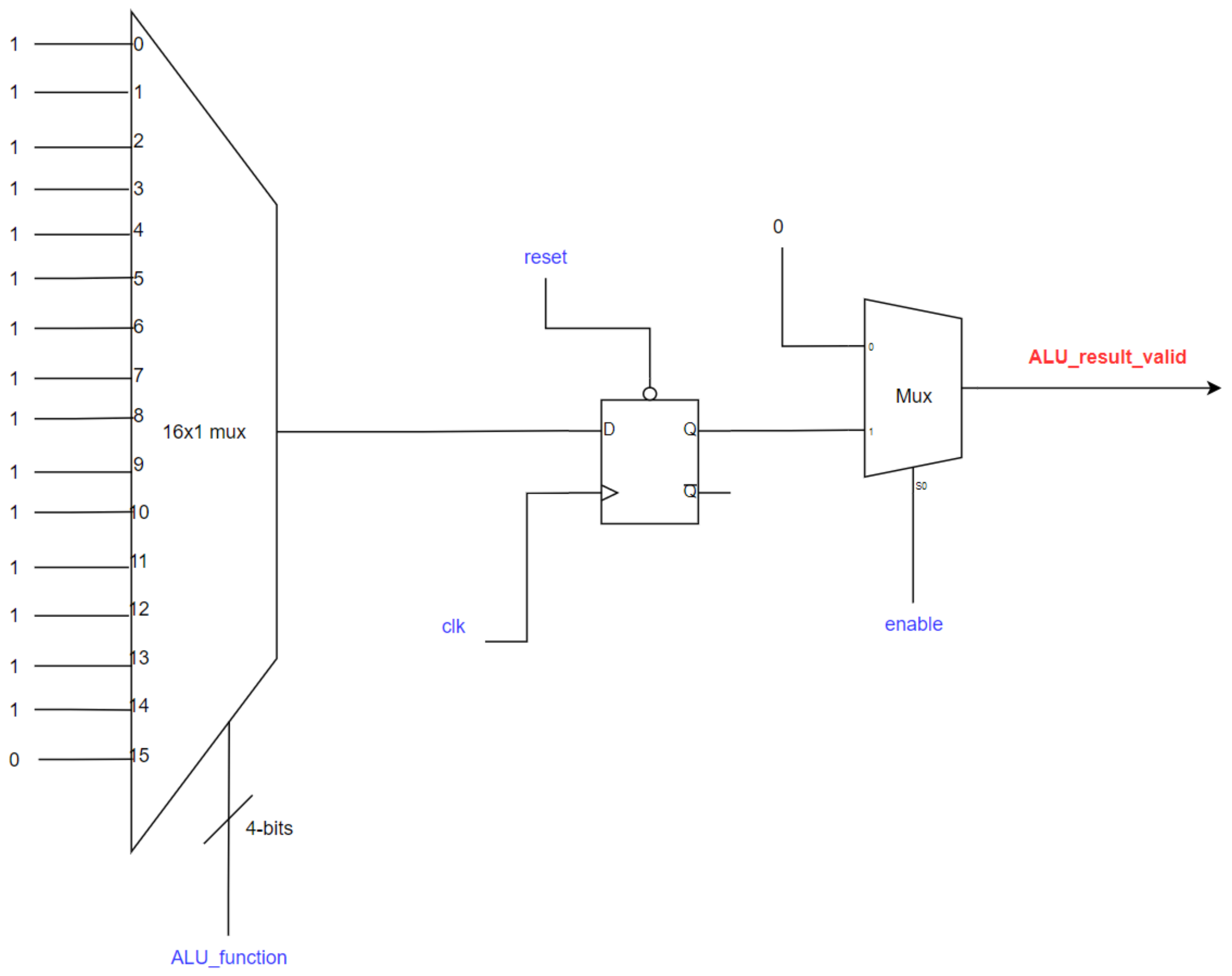
ALU

Logic Diagram

'ALU result' logic diagram:



'ALU result valid' logic diagram:



Port Decription

Port	Direction	Width	Description
clk	input	1	The ALU result is synchronized to this clock (reference clock).
reset	input	1	Global active low asynchronous reset after synchronization.
A	input	DATA_WIDTH (default value is 8)	ALU's first operand (it is connected to register_file[0]).
B	input	DATA_WIDTH (default value is 8)	ALU's second operand (it is connected to register_file[1]).
ALU_function	input	4	A binary code to determine the function of the ALU.
enable	input	1	ALU enable signal.
ALU_result_valid	output	1	A signal to indicate the ALU result is valid.
ALU_result	output	2 * DATA_WIDTH (default value is 16)	The result of the ALU.

Functional Verification

This module is verified through self-checking testbench in Modelsim. The testbench can be run using `run.tcl` script.

Register File

Port Decription

Port	Direction	Width	Description
clk	input	1	The read and write operations are synchronized to this clock (reference clock).
reset	input	1	Global active low asynchronous reset after synchronization.
address	input	$\log_2$(REGISTER_FILE_DEPTH) (default value is 4)	The address of the register to be read from or written to.
write_enable	input	1	A signal to enable the write operation.
write_data	input	DATA_WIDTH (default value is 8)	The data to be written in the given address.
read_enable	input	1	A signal to enable the read operation.
read_data_valid	output	1	A signal to indicate that the data on the 'read_data' bus is a valid data.
read_data	output	DATA_WIDTH (default value is 8)	The data read from the given address.
register0	output	DATA_WIDTH (default value is 8)	The first register in the register file, it stores the first operand of the ALU.
register1	output	DATA_WIDTH (default value is 8)	The second register in the register file, it stores the second operand of the ALU.
register2	output	DATA_WIDTH (default value is 8)	The third register in the register file, it stores the parity configuration (parity enable and parity type) of the UART.
register3	output	DATA_WIDTH (default value is 8)	The fourth register in the register file, it stores the value of the oversampling prescale used in the clock divider (it is the ratio between the clock frequency of the UART receiver and the clock frequency of the UART transmitter).

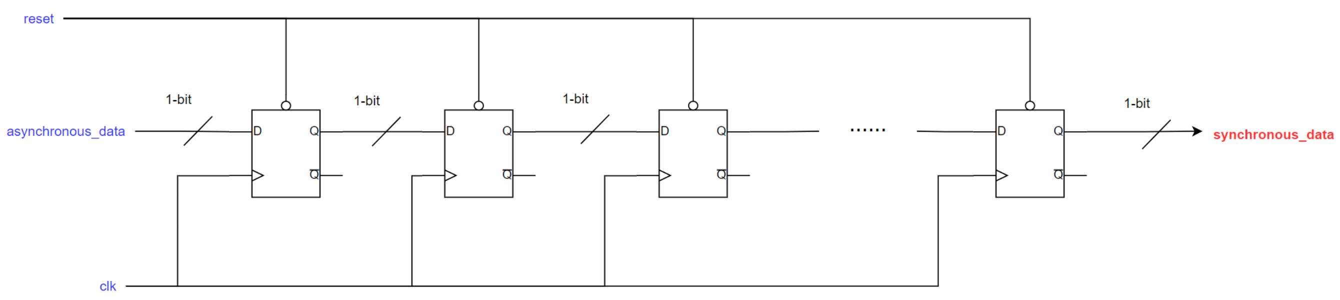
Functional Verification

This module is verified through self-checking testbench in Modelsim. The testbench can be run using `run.tcl` script.

Bus Synchronizer

It acts as a gray encoded bus synchronizer or a single bit synchronizer according to the value of the BUS_WIDTH parameter. It consists of multiple registers (single bit or multiple bits) connected in a cascaded scheme, and the number of stages is parametrized with a default value of 2.

Logic Diagram

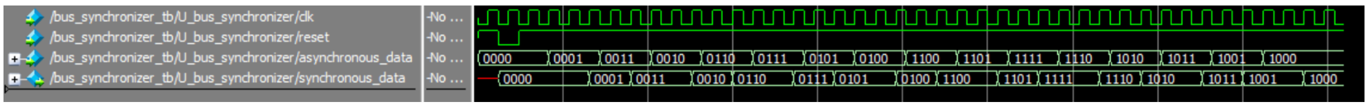


Port Description

Port	Direction	Width	Description
clk	input	1	The clock of the destination domain.
reset	input	1	Global active low asynchronous reset of the destination domain after synchronization.
asynchronous_data	input	BUS_WIDTH (default value is 1)	The data to be synchronized (it is sent from another asynchronous domain to the destination domain).
synchronous_data	output	BUS_WIDTH (default value is 1)	The data after synchronization to the destination domain.

Functional Verification

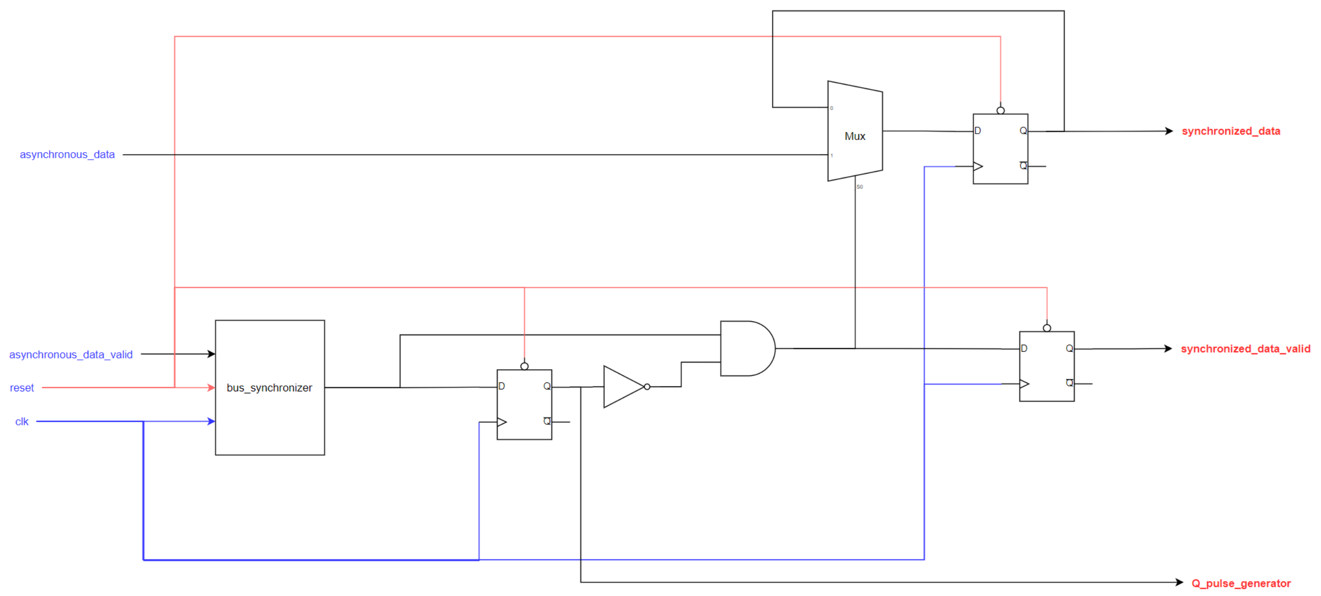
This module is verified through a Python script which generates all the possible binary gray codes of a given size, the testbench reads the gray codes from the external file and waveform simulation is performed in Modelsim. The simulation can be run using `run.tcl` script.


/bus_synchronizer_tb/U_bus_synchronizer/clk
/bus_synchronizer_tb/U_bus_synchronizer/reset
/bus_synchronizer_tb/U_bus_synchronizer/asynchronous_data
/bus_synchronizer_tb/U_bus_synchronizer/synchronous_data
0000 0001 0011 0010 0110 0111 0101 0100 1100 1101 1111 1110 1010 1011 1001 1000
0000 0001 0011 0010 0110 0111 0101 0100 1100 1101 1111 1110 1010 1011 1001 1000

Data Synchronizer

This module is used to synchronize any arbitrary bus by synchronizing its 'data valid' signal through a bit synchronizer then passing it through a pulse generator (flip-flop + NOT gate + AND gate) to produce a pulse whose width is the same as the width of destination domain. This pulse can be considered as a new 'data valid' signal synchronized to the destination domain and the data can be read safely from the 'asynchronous_data' port without the risk of entering metastability.

Logic Diagram



Port Description

Port	Direction	Width	Description
clk	input	1	The clock of the destination domain.
reset	input	1	Global active low asynchronous reset of the destination domain after synchronization.
asynchronous_data_valid	input	1	A signal to indicate that the data on the 'asynchronous_data' bus is valid.
asynchronous_data	input	BUS_WIDTH (default value is 8)	The data to be synchronized (it is sent from another asynchronous domain to the destination domain).
Q_pulse_generator	output	1	The output of the pulse generator register.
synchronous_data_valid	output	1	A signal to indicate that the synchronized data is valid.
synchronous_data	output	BUS_WIDTH (default value is 8)	The data after synchronization to the destination domain.

Functional Verification

This module is verified through self-checking testbench in Modelsim. The testbench can be run using `run.tcl` script.

System Controller

This moudule consists of the two submodules (UART Transmitter Controller, UART Receiver Controller) in which the transmitter controller sends an enable signal to the receiver controller to prevent it from processing a frame sent from the UART receiver while there is another frame being sent through the UART transmitter.

UART Transmitter Controller

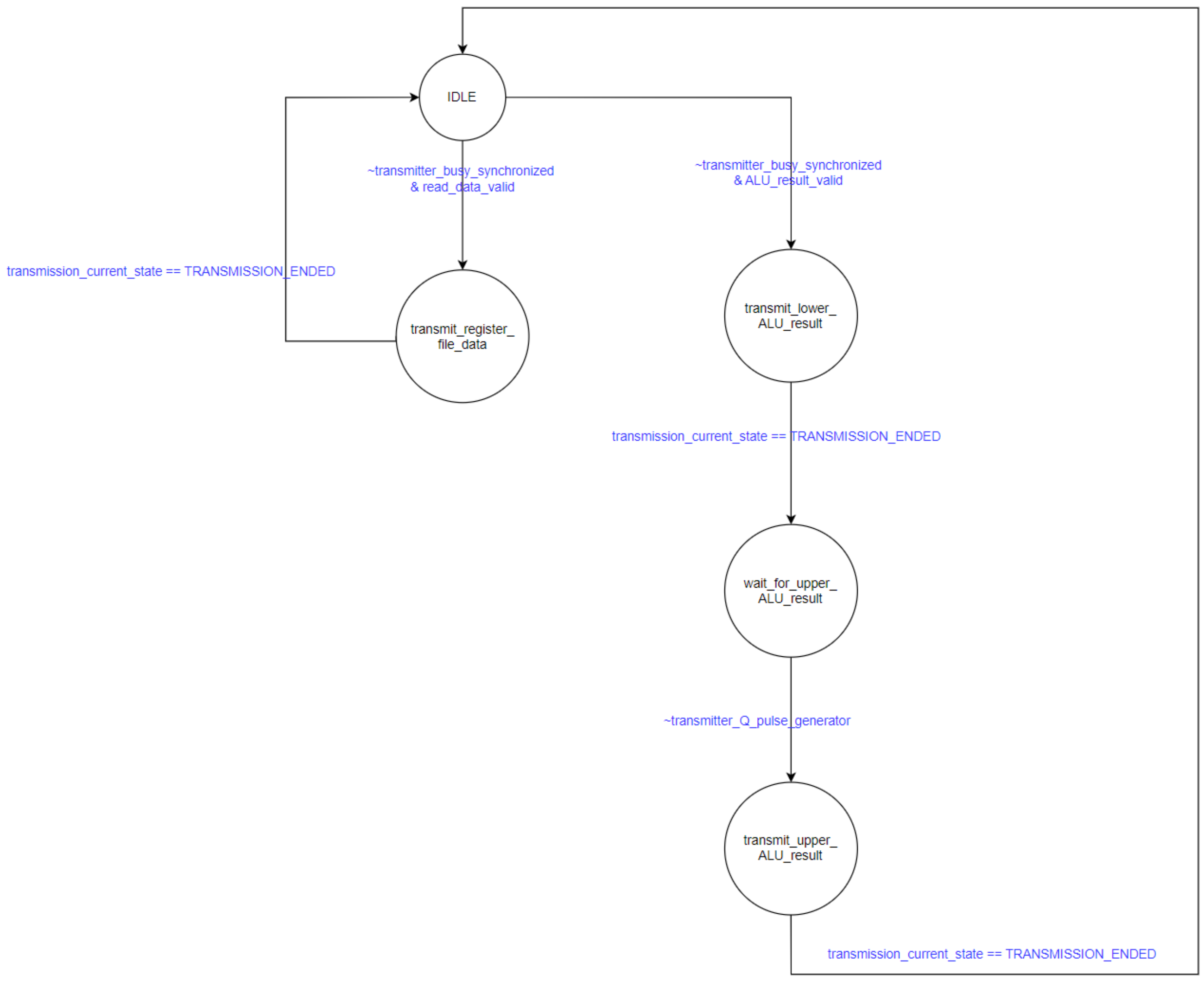
Finite State Machine (FSM)

This module consists of two FSMs:

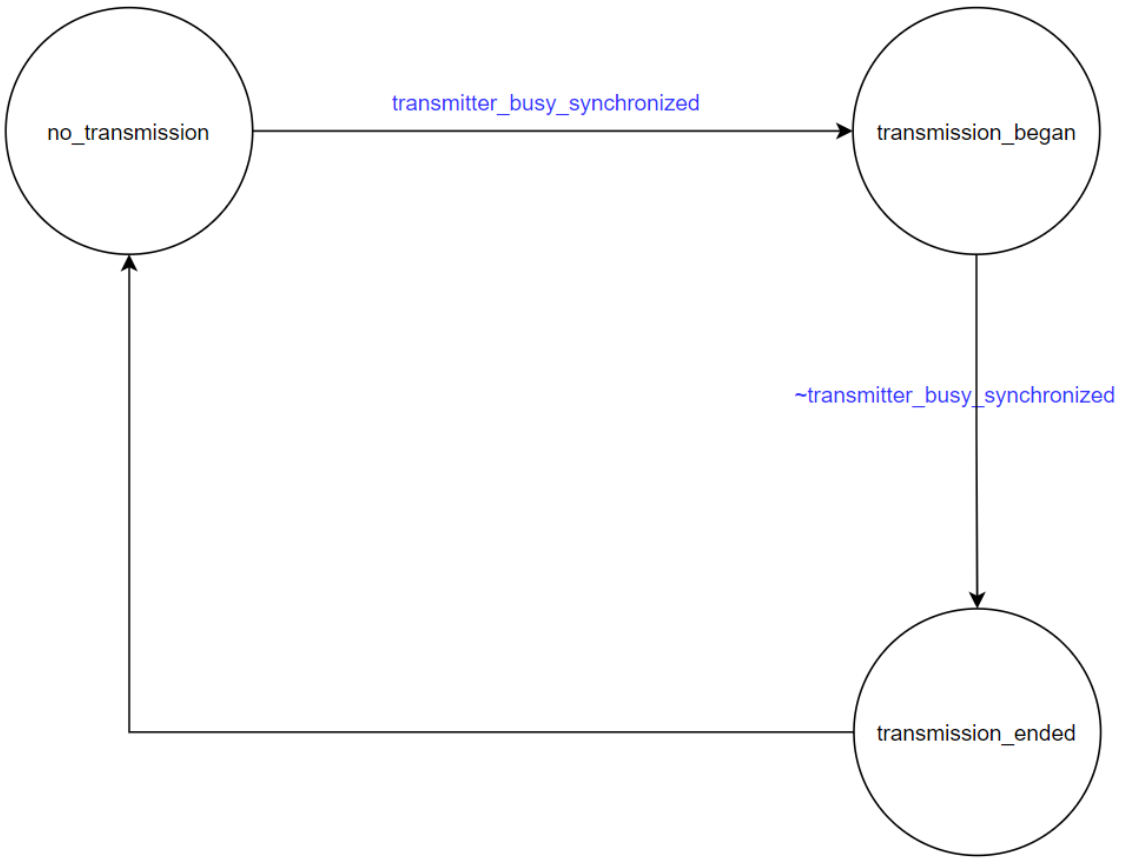
1. Main FSM: It controls the state which corresponds to which frame to send by the UART transmitter (e.g. TRANSMIT_REGISTER_FILE_DATA, TRANSMIT_LOWER_ALU_RESULT, TRANSMIT_UPPER_ALU_RESULT)
2. Transmission FSM: It is used to determine the status of the transmission (e.g. NO_TRANSMISSION, TRANSMISSION_BEGAN, TRANSMISSION_ENDED). It is used in the Main FSM to transition its state when the transmission has ended.

Main FSM: This FSM controls the following output ports according to the current state: transmitter_parallel_data_valid, transmitter_parallel_data, UART_receiver_controller_enable.

The "WAIT_FOR_UPPER_ALU_RESULT" state is used so that the "transmitter_parallel_data_valid" can be set to logic zero during that state, this is done so that the pulse generator can accept a new "data_valid" signal (this is because when the "transmitter_parallel_data_valid" is high (because of the transmission of the lower ALU result), the output of the NOT gate of the pulse generator is low and it won't change its value until the "transmitter_parallel_data_valid" becomes low. After that, the output of the NOT gate is high and a new "data_valid" signal can be sent). To exit "WAIT_FOR_UPPER_ALU_RESULT" state, the output of the pulse generator register "Q_pulse_generator" must be low (to ensure that the zero that was sent during the "WAIT_FOR_UPPER_ALU_RESULT" state has reached the pulse generator) and then the upper ALU result can be sent with a new "data_valid" signal. Note that the "Q_pulse_generator" is asynchronous to the reference clock domain (i.e. it is generated by the UART transmitter clock) so a bit synchronizer is used to synchronize it to the reference clock domain.


IDLE
~transmitter_busy_synchronized & read_data_valid
~transmitter_busy_synchronized & ALU_result_valid
transmission_current_state == TRANSMISSION_ENDED
transmit_register_file_data
transmit_lower_ALU_result
transmission_current_state == TRANSMISSION_ENDED
wait_for_upper_ALU_result
~transmitter_Q_pulse_generator
transmit_upper_ALU_result
transmission_current_state == TRANSMISSION_ENDED

Transmission FSM:



Note that if any omitted condition occurs, the current state won't change.

Port Description

Port	Direction	Width	Description
clk	input	1	Reference clock.
reset	input	1	Global active low asynchronous reset after synchronization.
ALU_result_valid	input	1	A signal to indicate the ALU result is valid.
ALU_result	input	2 * DATA_WIDTH (defualt value is 16)	The result of the ALU.
read_data_valid	input	1	The data read from the given address from the register file.
read_data	input	DATA_WDITH (default value is 8)	The data read from the given address from the register file.
transmitter_busy_synchronized	input	1	The UART transmitter busy signal after

			synchronization.
transmitter_Q_pulse_generator	input	1	The output signal of the pulse generator of the data synchronizer that synchronizes the UART transmitter data after being synchronized to the reference clock domain.
transmitter_parallel_data_valid	output	1	A signal to indicate that there is new data to be transmitted.
transmitter_parallel_data	output	DATA_WIDTH (default value is 8)	The data sent to the UART transmitter to transmit it serially.
UART_receiver_controller_enable	output	1	A signal to enable the operation of the controller, this signal is used to prevent the processing of frames while there is another frame being sent by the UART transmitter.

Functional Verification

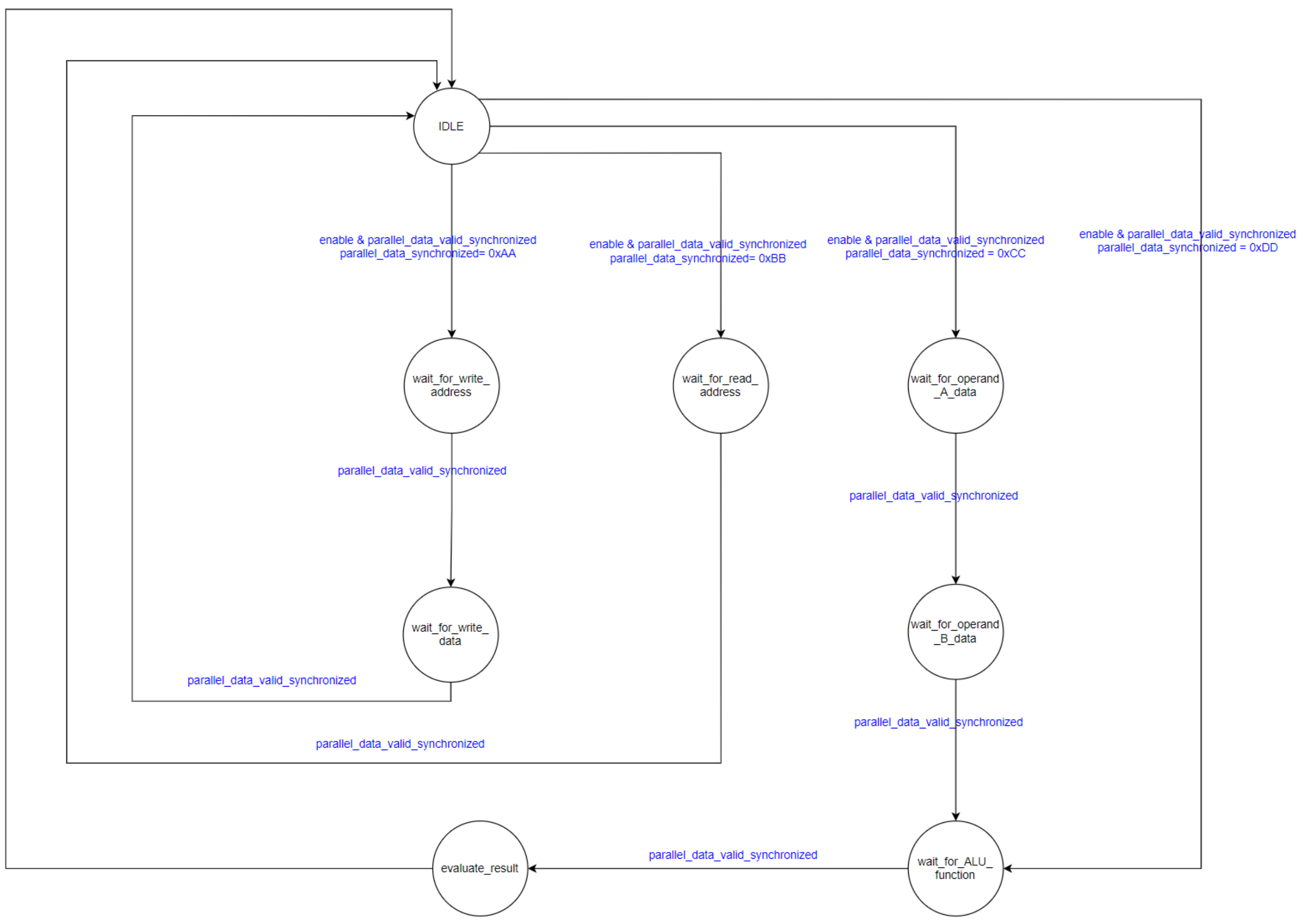
This module is verified through self-checking testbench in Modelsim. The testbench can be run using `run.tcl` script.

UART Receiver Controller

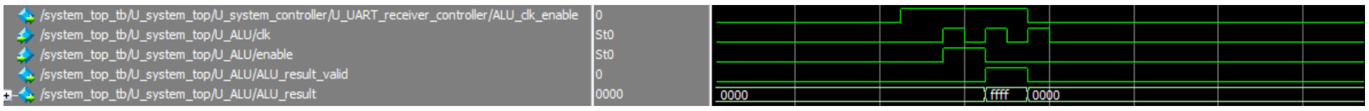
This module controls the ALU control signals (ALU_function, ALU_enable, ALU_clk_enable) and register file control signals (address, write_enable, write_data, read_enable) based on the received frames from the UART receiver (i.e. according to the command to be executed).

Finite State Machine (FSM)

The "EVALUATE_RESULT" state is a dummy state whose function is to delay the return to the "IDLE" state for one cycle so that "ALU_clk_enable" signal goes high for one cycle after the ALU result is evaluated.


IDLE
enable & parallel_data_valid_synchronized parallel_data_synchronized= 0xAA
enable & parallel_data_valid_synchronized parallel_data_synchronized= 0xBB
enable & parallel_data_valid_synchronized parallel_data_synchronized = 0xCC
enable & parallel_data_valid_synchronized parallel_data_synchronized = 0xDD
wait_for_write_address
wait_for_read_address
wait_for_operand_A_data
parallel_data_valid_synchronized
parallel_data_valid_synchronized
wait_for_write_data
wait_for_operand_B_data
parallel_data_valid_synchronized
parallel_data_valid_synchronized
parallel_data_valid_synchronized
evaluate_result
parallel_data_valid_synchronized
wait_for_ALU_function

The waveform after executing an ALU operation is shown in the following figure.


/system_top_tb/U_system_top/U_system_controller/U_UART_receiver_controller/ALU_clk_enable 0
/system_top_tb/U_system_top/U_ALU/clk St0
/system_top_tb/U_system_top/U_ALU/enable St0
/system_top_tb/U_system_top/U_ALU/ALU_result_valid 0
/system_top_tb/U_system_top/U_ALU/ALU_result 0000
0000
ffff
0000

Note that if any omitted condition occurs, the current state won't change.

Port Description

Port	Direction	Width	Description
clk	input	1	Reference clock.
reset	input	1	Global active low asynchronous reset after synchronization.
enable	input	1	A signal to enable the operation of the controller, this signal is used to prevent the processing of frames while there is another frame being sent by the UART transmitter.
parallel_data_valid_synchronized	input	1	A synchrnoized signal to

			indicate that there exist new data received by the UART receiver.
parallel_data_synchronized	output	DATA_WIDTH (default value is 8)	The data received by the UART receiver.
ALU_function	output	4	A binary code to determine the function of the ALU.
ALU_enable	output	1	A signal to enable the operation of the ALU.
ALU_clk_enable	output	1	A signal to enable the clock gating cell which outputs the ALU clock.
address	output	$\log_2$(REGISTER_FILE_DEPTH) (defult value is 4)	The address of the register to be read from or written to.
write_enable	output	1	A signal to enable the read operation from the register file.
write_data	output	DATA_WIDTH (default value is 8)	The data to be written in the given address in the register file.
read_enable	output	1	A signal to enable the read operation in the register file.

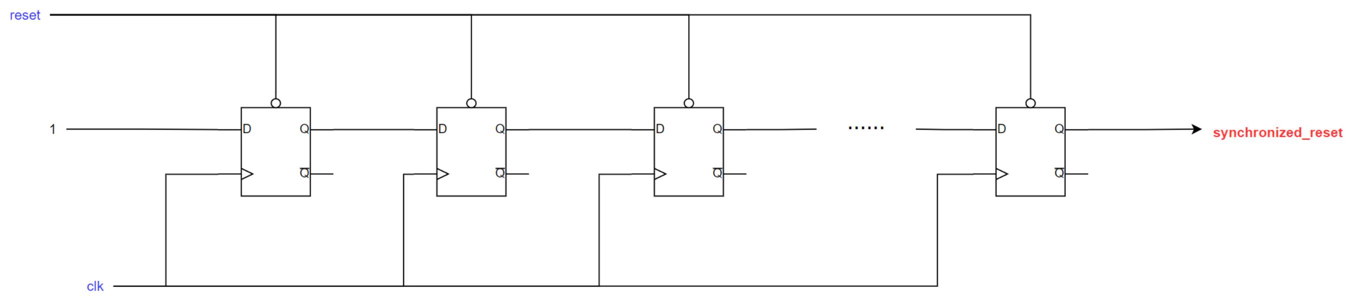
Functional Verification

This module is verified through self-checking testbench in Modelsim. The testbench can be run using `run.tcl` script.

Reset Synchronizer

It is used to synchronize a global reset signal to different clock domains. It consists of multiple flip-flops connected in a cascaded scheme, and the number of stages is parametrized with a default value of 2.

Logic Diagram

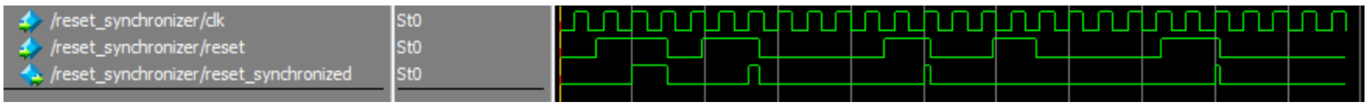


Port Decription

Port	Direction	Width	Description
clk	input	1	The clock of the destination domain.
reset	input	1	Unsynchronized global active low reset.
reset_synchronized	output	1	The reset signal after synchronization to the destination domain.

Functional Verification

This module is verified through logic simulation in Modelsim. The simulation can be run using `run.tcl` script.


/reset_synchronizer/clk St0
/reset_synchronizer/reset St0
/reset_synchronizer/reset_synchronized St0

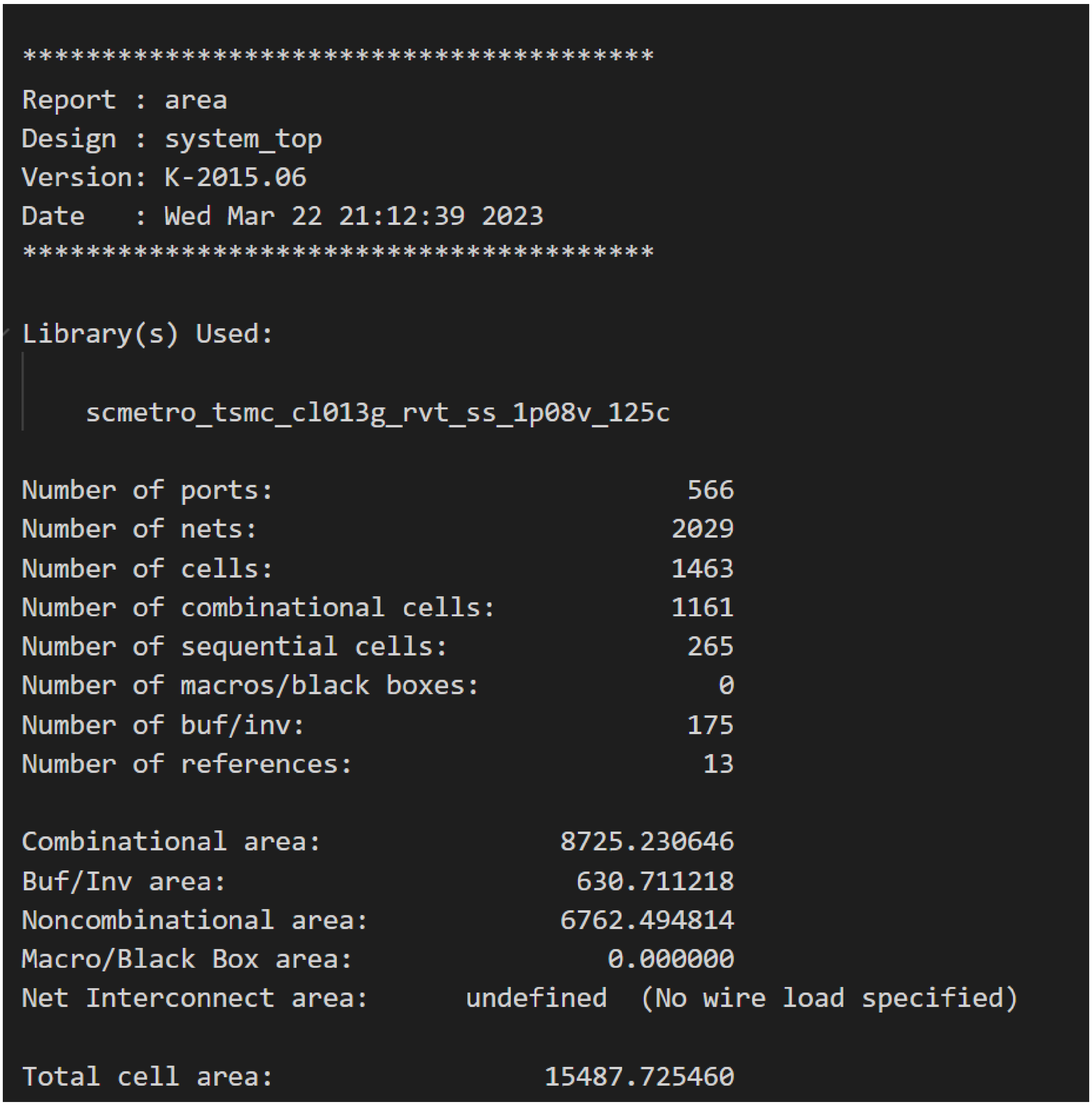
Logic Synthesis

The whole system is synthesized using Synopsys Design Compiler and "TSMC 130nm CL013G-FSG Process 1.2V Metro^{TM} v1.0" Standard Cell Library. There are 3 scripts that are used in the synthesis process:

1. 'constraints.tcl': It defines all clock source using 'create_clock' and 'create_generated_clock', defines asynchronous clock groups using 'set_clock_groups -asynchronous', sets input and output delays, sets the output load on the output ports, and sets the different operating conditions to be used in timing analysis (slow-slow library is used for setup analysis, fast-fast library is used for hold analysis).
2. 'logic_synthesis_script.tcl': It places the ICG instead of the RTL module, reads libraries, reads RTL modules, links and compiles the design, generates the reports, netlist, and SDF file, and replaces the ICG with the RTL module again (to be used in simulations).
3. 'run.sh': It runs the 'logic_synthesis_script.tcl' and produces the log file which contains all the steps of the synthesis and their outputs and cleans the directory from temporary files after the synthesis is done.

Reports

Area

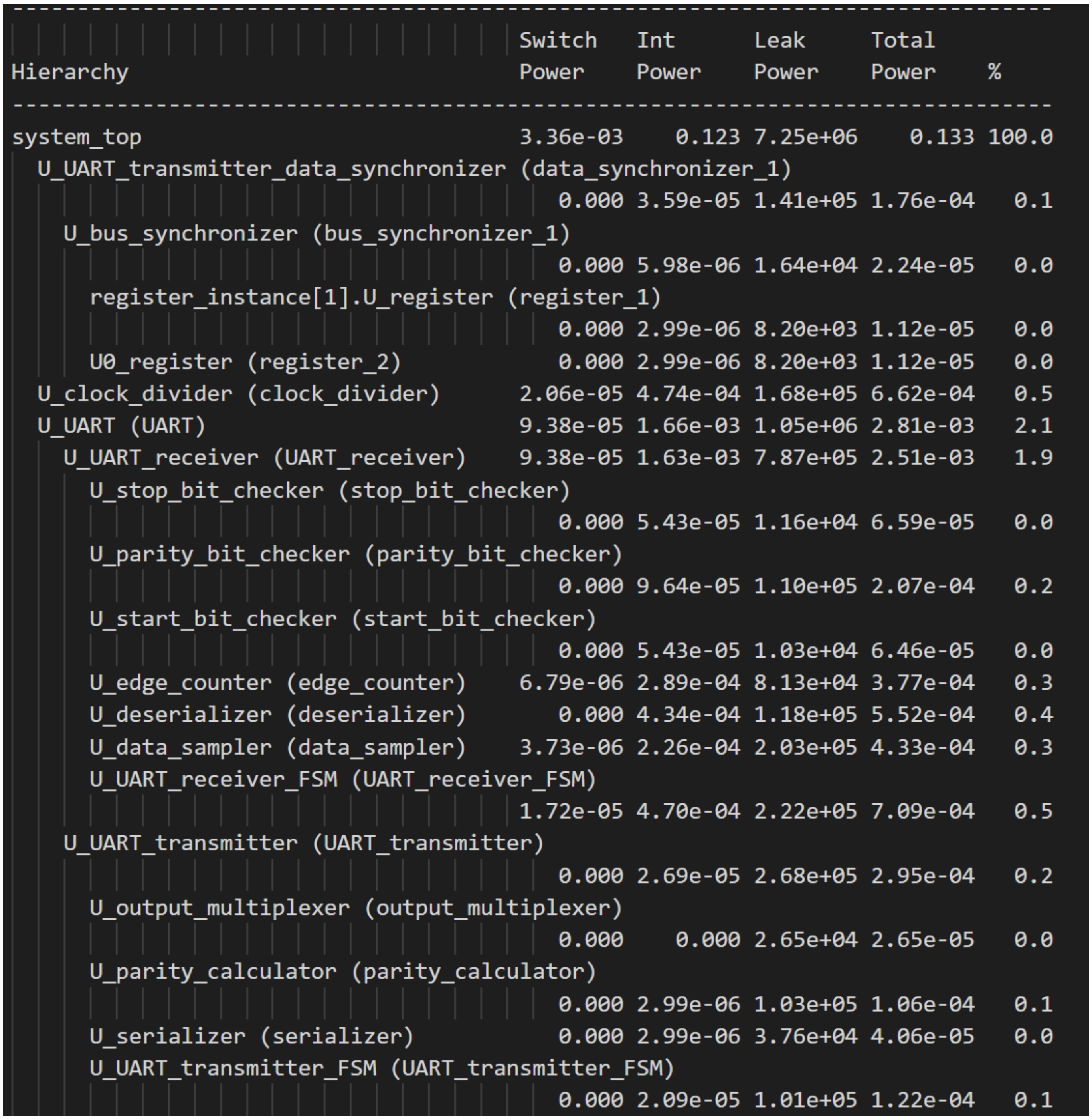


```
****************************************
Report : area
Design : system_top
Version: K-2015.06
Date   : Wed Mar 22 21:12:39 2023
****************************************

Library(s) Used:

    scmetro_tsmc_cl013g_rvt_ss_1p08v_125c

Number of ports:                          566
Number of nets:                          2029
Number of cells:                         1463
Number of combinational cells:           1161
Number of sequential cells:               265
Number of macros/black boxes:               0
Number of buf/inv:                        175
Number of references:                      13

Combinational area:               8725.230646
Buf/Inv area:                      630.711218
Noncombinational area:            6762.494814
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                 15487.725460
```

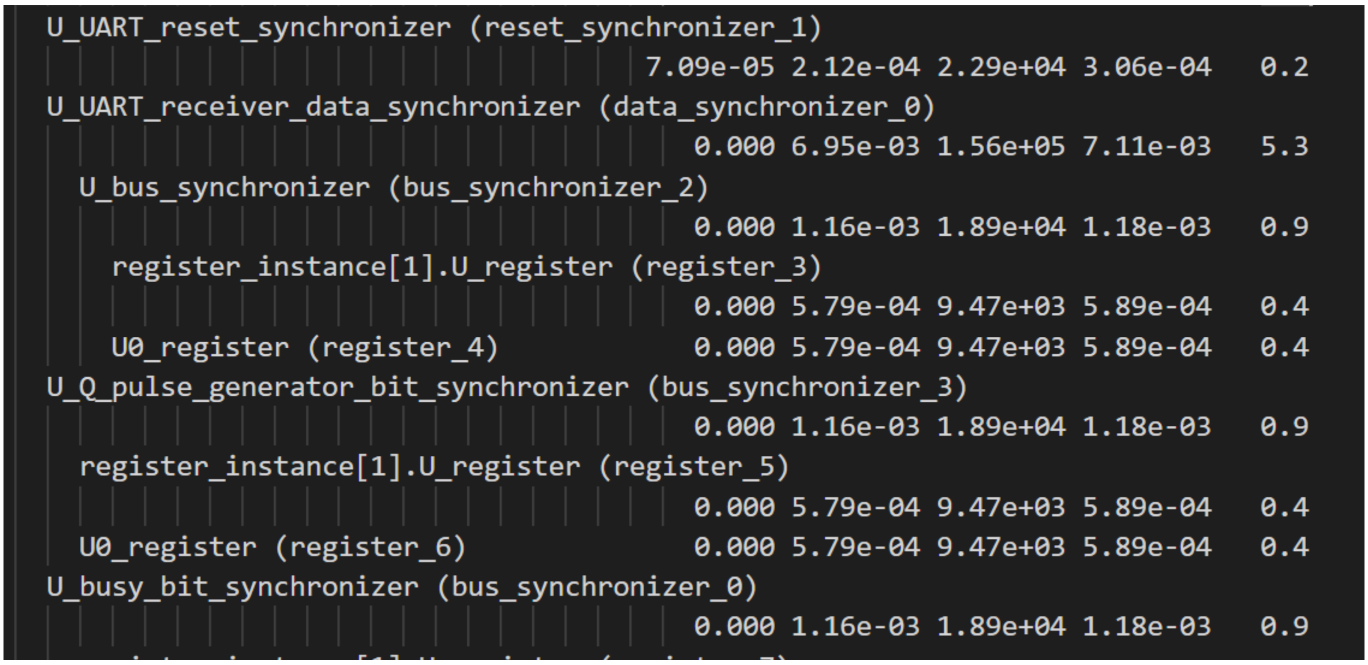
Clocks

```
****************************************
Report : clocks
Design : system_top
Version: K-2015.06
Date   : Wed Mar 22 21:12:39 2023
****************************************

Attributes:
    d - dont_touch_network
    f - fix_hold
    p - propagated_clock
    G - generated_clock
    g - lib_generated_clock

Clock          Period   Waveform            Attrs     Sources
--------------------------------------------------------------------------------
ALU_CLK         25.00   {12 25}             G         {U_clock_gating_cell/gated_clock}
REFERENCE_CLK   25.00   {12 25}                       {reference_clk}
UART_CLK       271.00   {0 135}                       {UART_clk}
UART_TRANSMITTER_CLK
              8672.00   {0 4336}            G         {U_clock_divider/output_clk}
--------------------------------------------------------------------------------

Generated     Master         Generated      Master         Waveform
Clock         Source         Source         Clock          Modification
--------------------------------------------------------------------------------
ALU_CLK       reference_clk  {U_clock_gating_cell/gated_clock}
                                            REFERENCE_CLK  divide_by(1)
UART_TRANSMITTER_CLK
              UART_clk       {U_clock_divider/output_clk}
                                            UART_CLK       divide_by(32)
--------------------------------------------------------------------------------
```

Power



```
--------------------------------------------------------------------------------
                                       Switch   Int      Leak     Total
Hierarchy                              Power    Power    Power    Power    %
--------------------------------------------------------------------------------
system_top                             3.36e-03    0.123 7.25e+06    0.133 100.0
  U_UART_transmitter_data_synchronizer (data_synchronizer_1)
                                          0.000 3.59e-05 1.41e+05 1.76e-04   0.1
    U_bus_synchronizer (bus_synchronizer_1)
                                          0.000 5.98e-06 1.64e+04 2.24e-05   0.0
      register_instance[1].U_register (register_1)
                                          0.000 2.99e-06 8.20e+03 1.12e-05   0.0
      U0_register (register_2)            0.000 2.99e-06 8.20e+03 1.12e-05   0.0
  U_clock_divider (clock_divider)      2.06e-05 4.74e-04 1.68e+05 6.62e-04   0.5
  U_UART (UART)                        9.38e-05 1.66e-03 1.05e+06 2.81e-03   2.1
    U_UART_receiver (UART_receiver)    9.38e-05 1.63e-03 7.87e+05 2.51e-03   1.9
      U_stop_bit_checker (stop_bit_checker)
                                          0.000 5.43e-05 1.16e+04 6.59e-05   0.0
      U_parity_bit_checker (parity_bit_checker)
                                          0.000 9.64e-05 1.10e+05 2.07e-04   0.2
      U_start_bit_checker (start_bit_checker)
                                          0.000 5.43e-05 1.03e+04 6.46e-05   0.0
      U_edge_counter (edge_counter)    6.79e-06 2.89e-04 8.13e+04 3.77e-04   0.3
      U_deserializer (deserializer)       0.000 4.34e-04 1.18e+05 5.52e-04   0.4
      U_data_sampler (data_sampler)    3.73e-06 2.26e-04 2.03e+05 4.33e-04   0.3
      U_UART_receiver_FSM (UART_receiver_FSM)
                                       1.72e-05 4.70e-04 2.22e+05 7.09e-04   0.5
    U_UART_transmitter (UART_transmitter)
                                          0.000 2.69e-05 2.68e+05 2.95e-04   0.2
      U_output_multiplexer (output_multiplexer)
                                          0.000    0.000 2.65e+04 2.65e-05   0.0
      U_parity_calculator (parity_calculator)
                                          0.000 2.99e-06 1.03e+05 1.06e-04   0.1
      U_serializer (serializer)           0.000 2.99e-06 3.76e+04 4.06e-05   0.0
      U_UART_transmitter_FSM (UART_transmitter_FSM)
                                          0.000 2.09e-05 1.01e+05 1.22e-04   0.1
```



```
  U_UART_reset_synchronizer (reset_synchronizer_1)
                                       7.09e-05 2.12e-04 2.29e+04 3.06e-04   0.2
  U_UART_receiver_data_synchronizer (data_synchronizer_0)
                                          0.000 6.95e-03 1.56e+05 7.11e-03   5.3
    U_bus_synchronizer (bus_synchronizer_2)
                                          0.000 1.16e-03 1.89e+04 1.18e-03   0.9
      register_instance[1].U_register (register_3)
                                          0.000 5.79e-04 9.47e+03 5.89e-04   0.4
      U0_register (register_4)            0.000 5.79e-04 9.47e+03 5.89e-04   0.4
  U_Q_pulse_generator_bit_synchronizer (bus_synchronizer_3)
                                          0.000 1.16e-03 1.89e+04 1.18e-03   0.9
    register_instance[1].U_register (register_5)
                                          0.000 5.79e-04 9.47e+03 5.89e-04   0.4
    U0_register (register_6)              0.000 5.79e-04 9.47e+03 5.89e-04   0.4
  U_busy_bit_synchronizer (bus_synchronizer_0)
                                          0.000 1.16e-03 1.89e+04 1.18e-03   0.9
    register_instance[1].U_register (register_7)
                                   0.000 5.79e-04 9.47e+03 5.89e-04   0.4
    U0_register (register_0)       0.000 5.79e-04 9.47e+03 5.89e-04   0.4
  U_register_file (register_file)      1.28e-03 7.95e-02 2.48e+06 8.33e-02  62.5
  U_ALU (ALU)                      0.000 9.87e-03 2.54e+06 1.24e-02   9.3
  U_clock_gating_cell (clock_gating_cell)
                                1.18e-03 1.63e-03 1.49e+04 2.82e-03   2.1
  U_system_controller (system_controller)
                                3.02e-04 1.80e-02 6.03e+05 1.89e-02  14.2
    U_UART_receiver_controller (UART_receiver_controller)
                                   0.000 5.21e-03 2.04e+05 5.42e-03   4.1
    U_UART_transmitter_controller (UART_transmitter_controller)
                                   0.000 1.28e-02 3.95e+05 1.32e-02   9.9
  U_reference_reset_synchronizer (reset_synchronizer_0)
                                1.18e-05 1.91e-03 2.78e+04 1.95e-03   1.5
```

Timing reports (setup and hold) can be found at "logic_synthesis/logic_synthesis_output/reports"

Post - Logic Synthesis Formal Verification

Formal verification takes the original RTL modules (golden RTL files) and the netlist generated fom the synthesis process and performs functional equivalence checking and reports whether the two systems (RTL and netlist) have the same functionality or not. The formal verification is run using Synopsys Formality. There are 2 scripts that are used in the formal verification process:

1. 'formal_verification_script.tcl': It places the ICG instead of the RTL module, reads libraries, reads golden RTL modules, reads the netlist generated from the synthesis process, compares the two designs, and replaces the ICG with the RTL module again (to be used in simulations).
2. 'run.sh': It runs the 'formal_verification_script.tcl' and produces the log file which contains all the steps of the formal verification and their outputs and cleans the directory from temporary files after the verification is done.

Verification Results

```
******************************** Verification Results ********************************
Verification SUCCEEDED
   ATTENTION: synopsys_auto_setup mode was enabled.
              See Synopsys Auto Setup Summary for details.
-----------------------------------------------------------
 Reference design: reference:/WORK/system_top
 Implementation design: implementation:/WORK/system_top
 264 Passing compare points
---------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
---------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0       3     260       1     264
Failing (not equivalent)       0       0       0       0       0       0       0       0
***************************************************************************************
```

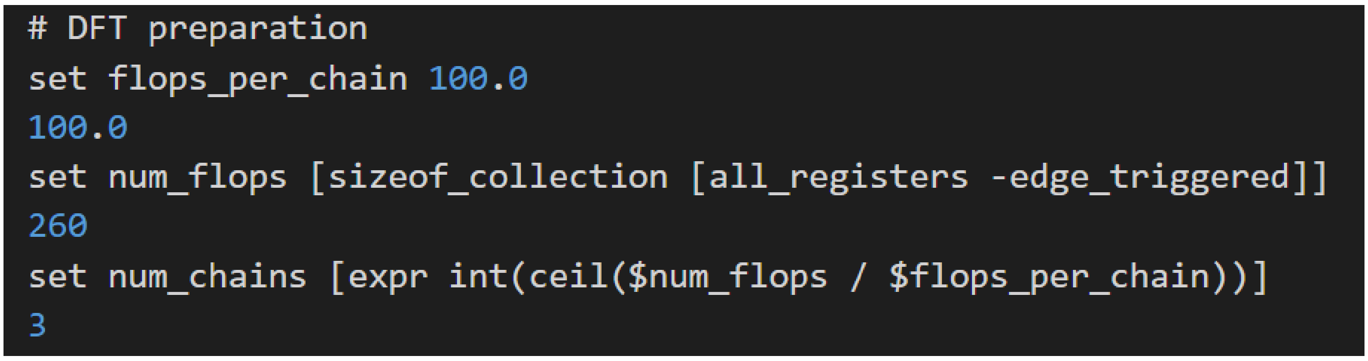
Design For Testability (DFT)

DFT is used to insert additional logic (not for functional mode) to ensure that chip is free of manufacturing errors. The main concept is to create scan chains consisting of all registers in the system to be able to observe and control most of the nodes of the circuit (i.e. achieve high coverage). After DFT insertion, all the flip-flops (except shift registers) are replaced with scan flip-flops.

DFT ports:

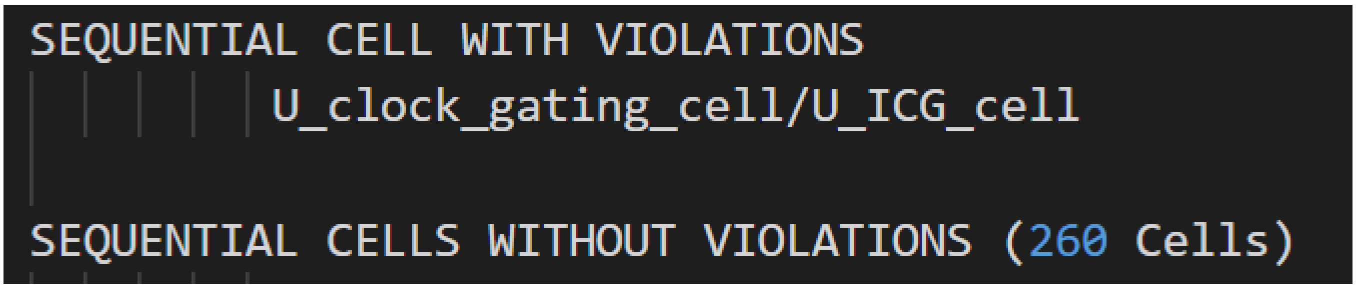
- scan_clk: The clock which the system uses in test mode
- scan_reset: The reset which the system uses in the test mode
- SE: Scan enable signal, when SE = 1: all the registers in the scan chain acts as asingle shift register, when SE = 0: The output of the register passes through the combinational logic following it (this is used to to test nodes in the circuit). SE is connected to all the SE pins of all the scan flops.
- test_mode: A signal to indicate whether the chip is operating in the functional mode or the test mode.
- SI: The input to the scan chain. The width of this port is the number of the scan chains
- SO: The output from the scan chain. The width of this port is the number of the scan chains

Number of scan chains: These results are produced after the logic synthesis indicating that there should exist 3 scan chains in the system if each chain has 100 flip-flops.



```
# DFT preparation
set flops_per_chain 100.0
100.0
set num_flops [sizeof_collection [all_registers -edge_triggered]]
260
set num_chains [expr int(ceil($num_flops / $flops_per_chain))]
3
```

The logic of the scan_clk and scan_reset is manually inserted by using 2x1 muxes for all the clocks and resets in the system. The clock gating cell is bypassed by connecting its enable pin with (test_mode | enable) (i.e. the clock gating cell will be always enabled in the test mode). This leads to a violation on this cell because it is not controllable.



```
SEQUENTIAL CELL WITH VIOLATIONS
│ │ │ │ │ U_clock_gating_cell/U_ICG_cell
│

SEQUENTIAL CELLS WITHOUT VIOLATIONS (260 Cells)
```

There are 3 modes of operation after the DFT logic is inserted:

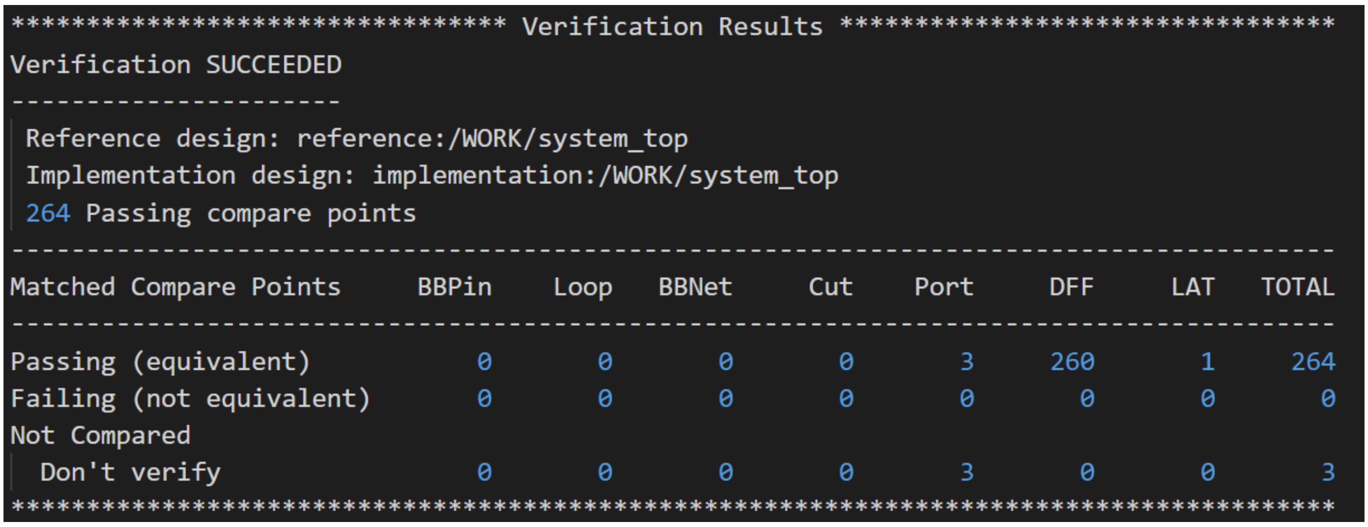
1. Functional mode: test_mode = 0, SE = 0
2. Test scan mode: test_mode = 1, SE = 1
3. Test capture mode: test_mode = 1, SE = 0

The whole system is synthesized and DFT insertion is performed using Synopsys DFT Compiler and "TSMC 130nm CL013G-FSG Process 1.2V Metro^{TM} v1.0" Standard Cell Library. The synthesis and DFT insertion is repeated 3 times (once for each mode of operation) to ensure that there is no violations in any mode. There are 3 scripts that are used in the DFT insertion process:

1. 'constraints.tcl': Same constraints of the logic synthesis process but it additionaly defines the scan clock to be used in timing analysis
2. 'DFT_script.tcl': Same operations done in the logic synthesis process but it additionaly defines the DFT ports, identify the shift registers in the design so that they are not replaced with scan flip-flops, and insert DFT logic.
3. 'run.sh': It runs the 'DFT_script.tcl' 3 times by using the 'change_mode.py' script which changes the mode of operation automatically and produces the log file which contains all the steps of the synthesis and their outputs and cleans the directory from temporary files after the DFT insertion is done.

Post - DFT Formal Verification

Verification Results



```
******************************** Verification Results ********************************
Verification SUCCEEDED
----------------------
 Reference design: reference:/WORK/system_top
 Implementation design: implementation:/WORK/system_top
 264 Passing compare points
--------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
--------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0       3     260       1     264
Failing (not equivalent)       0       0       0       0       0       0       0       0
Not Compared
  Don't verify                 0       0       0       0       3       0       0       3
**************************************************************************************
```

There exist 3 don't compare points which are the 3 output pins of the SO port, they are not verified because they doesn't exeist in the golden RTL files (i.e. their logic was automatically inserted by the DFT tool).

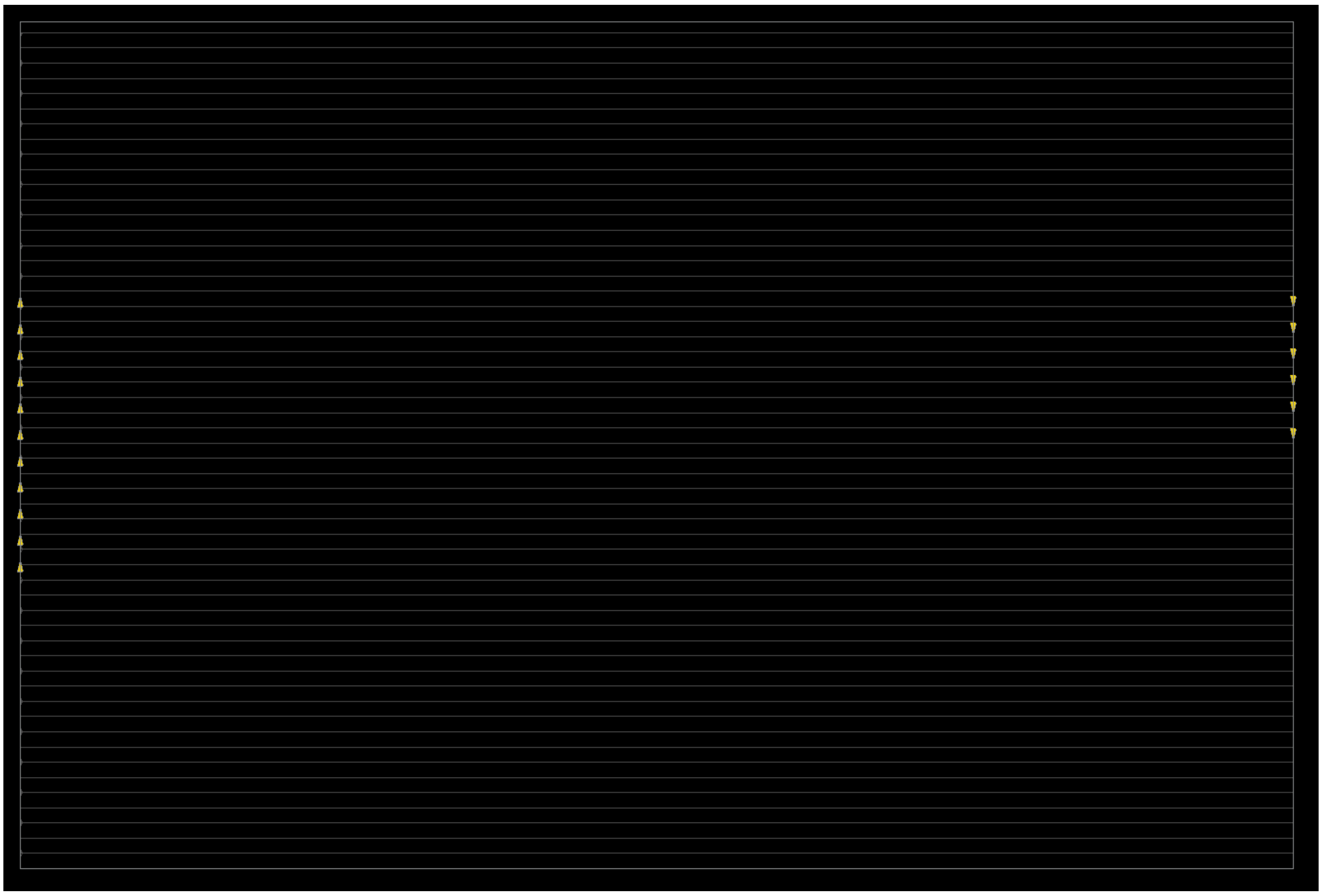
Physical Design

Import Design

In this step: the netlist generated from the DFT process, standard cells' lef file, technology lef file (6 metal layers are used), and system_top lef file (which specifies the layout of the whole system) are loaded. The MMMC (multi-mode multi-corner) constraints file is also loaded, this file defines the three modes of operations of the system (as explained in the DFT section) and also defines all the corners to be used in static timing analysis (the corners are defined by the fast-fast library and slow-slow library).

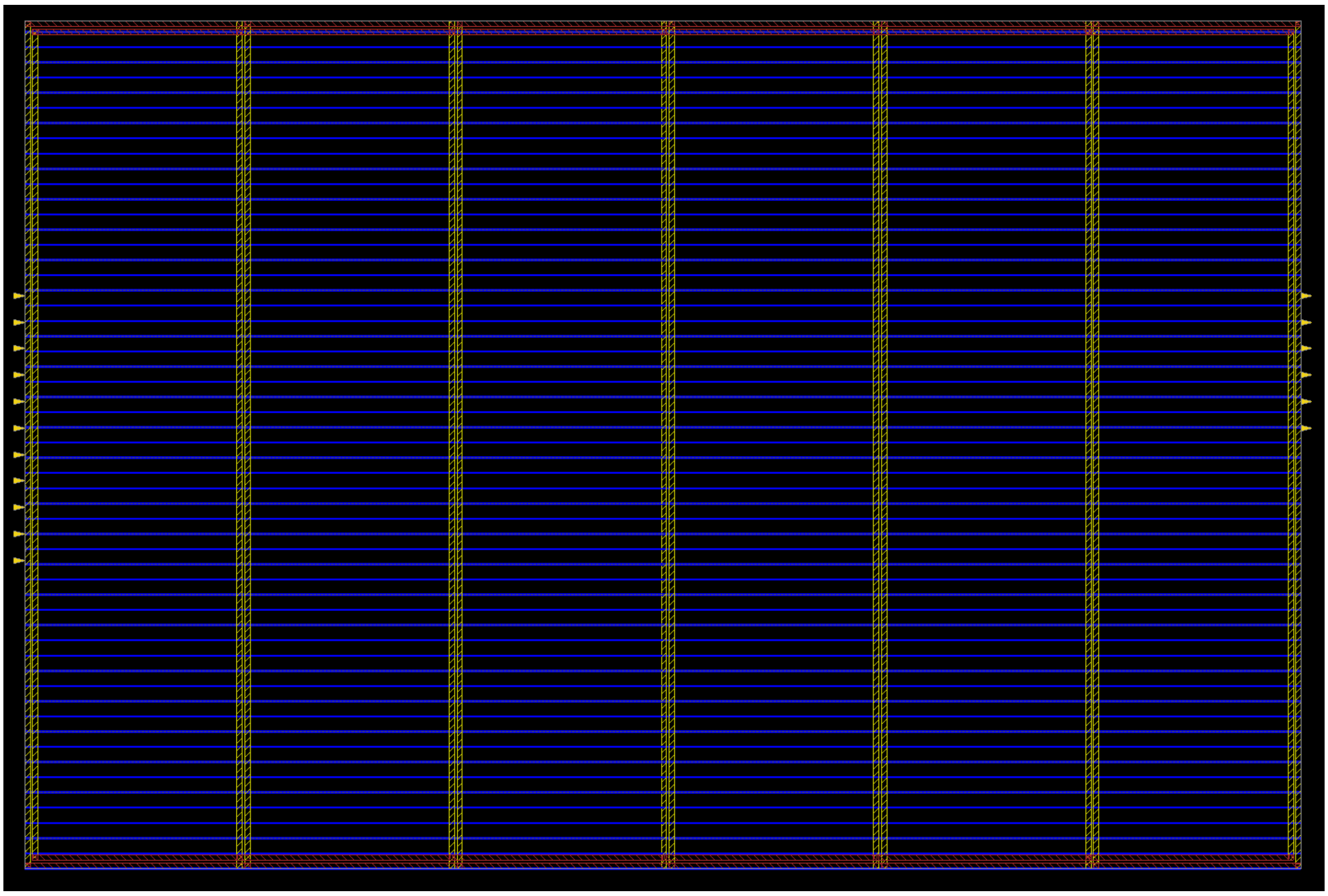
Floor Planning

The chip size is 240.67x160.0 µm^2.



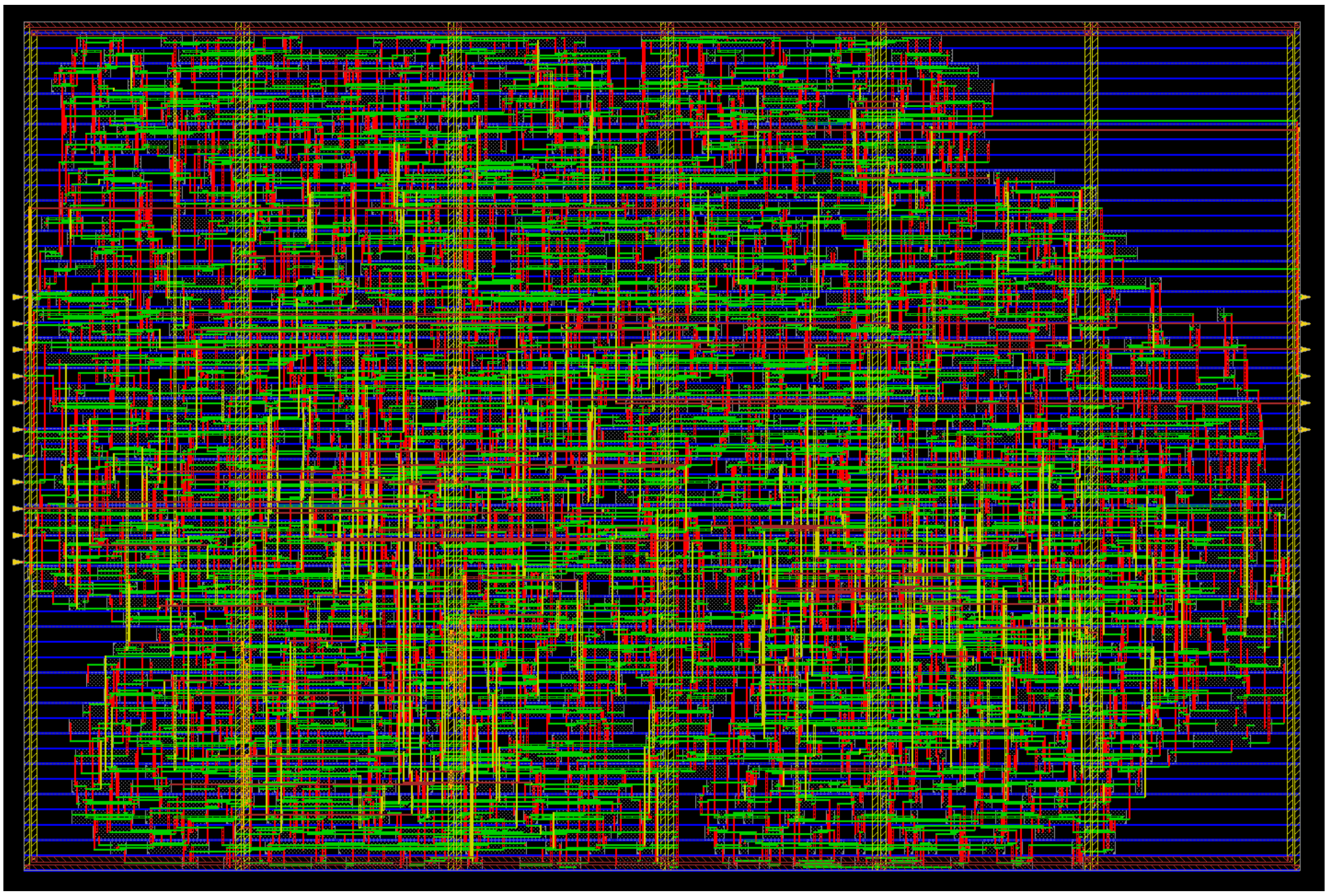
Power Planning

The power planning step is performed to reduce the IR drop and minimize the effect of electromigration. In this step: power rings, power stripes, and power rails are inserted.



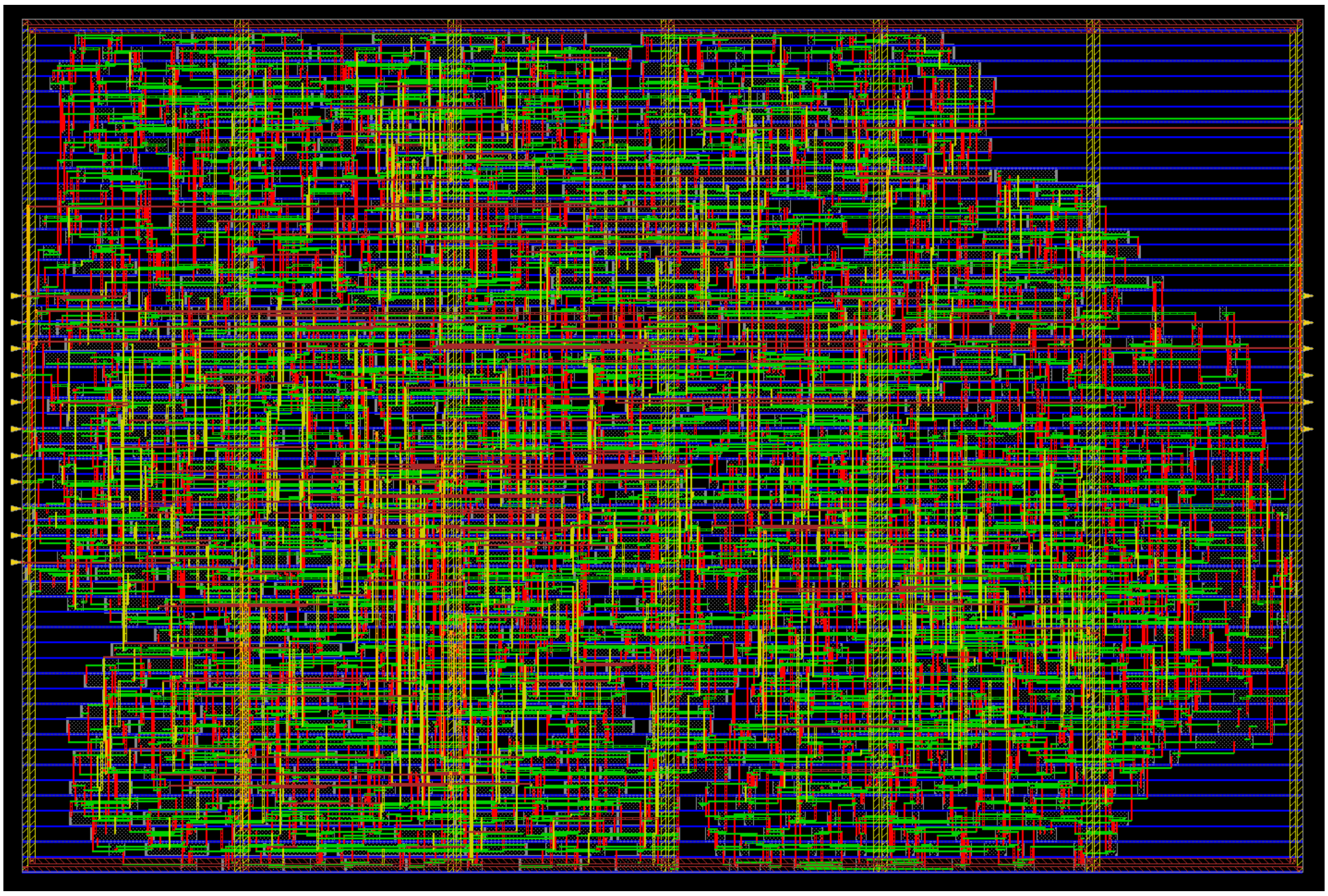
Placement

Place the stanard cells and optimize if there is any timing violation.



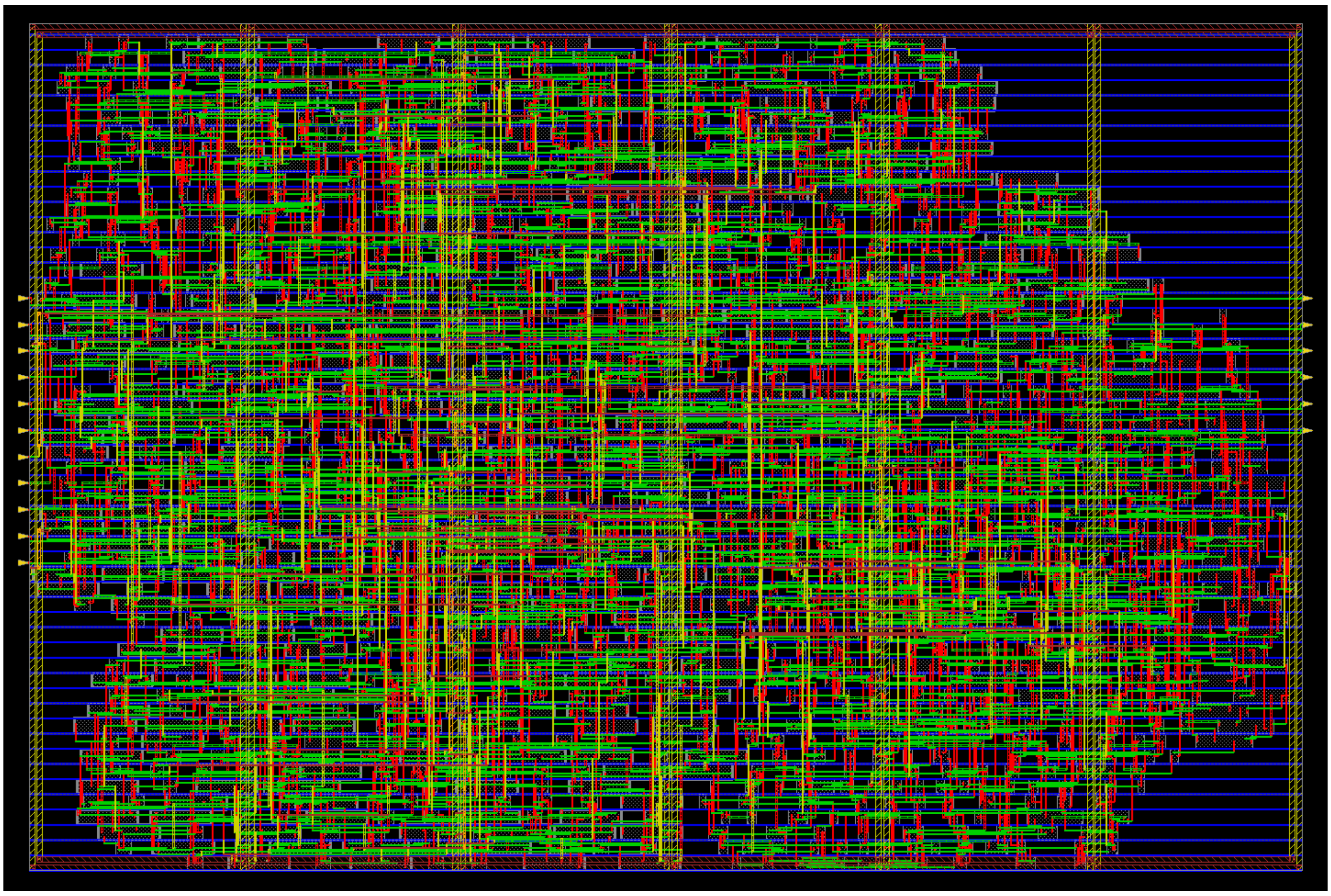
Clock Tree Synthesis (CTS)

Build the clock tree to minimize the clock skew between registers and optimize if there is any timing violation.



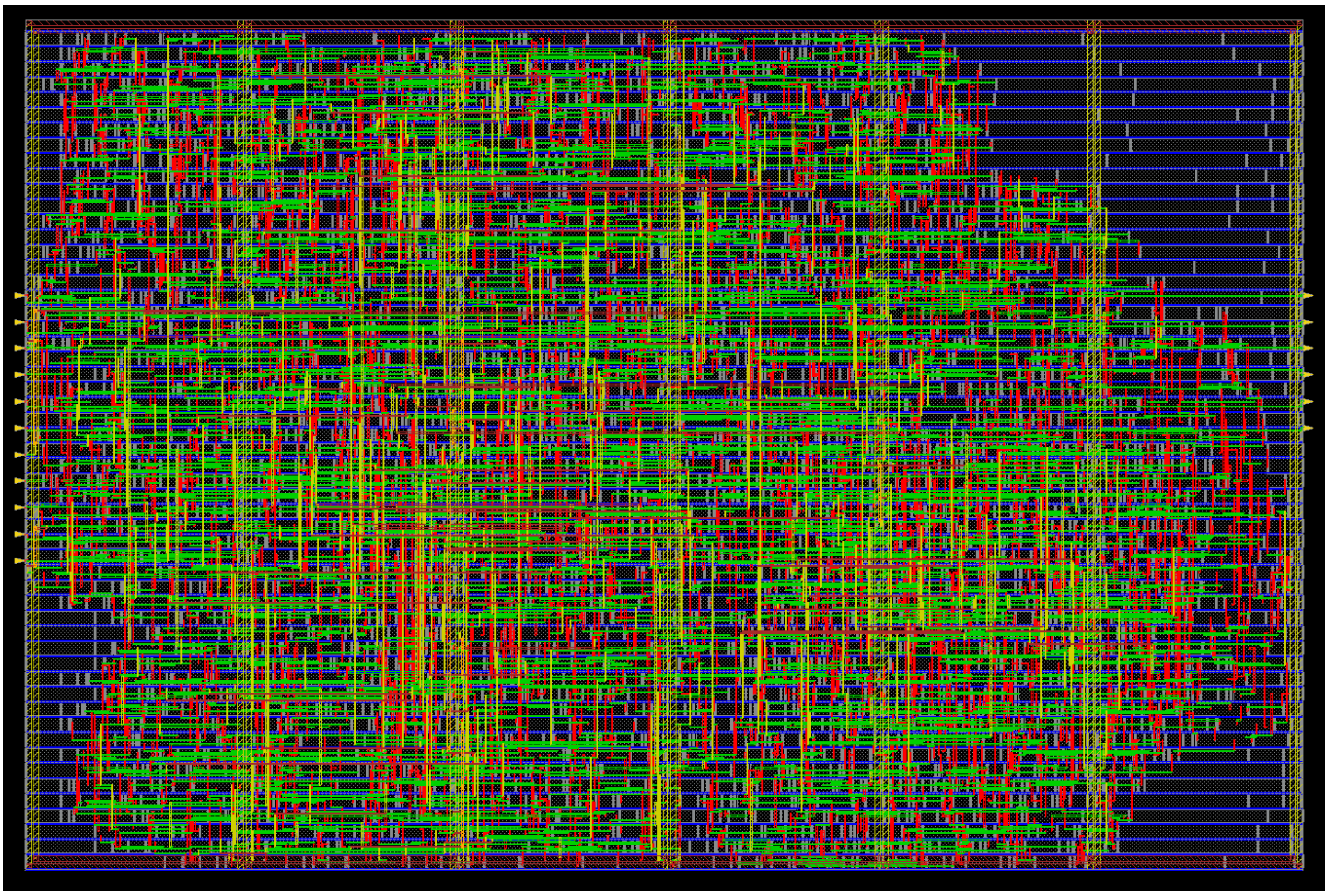
Routing

Route the standard cells and optimize if there is any timing violation.



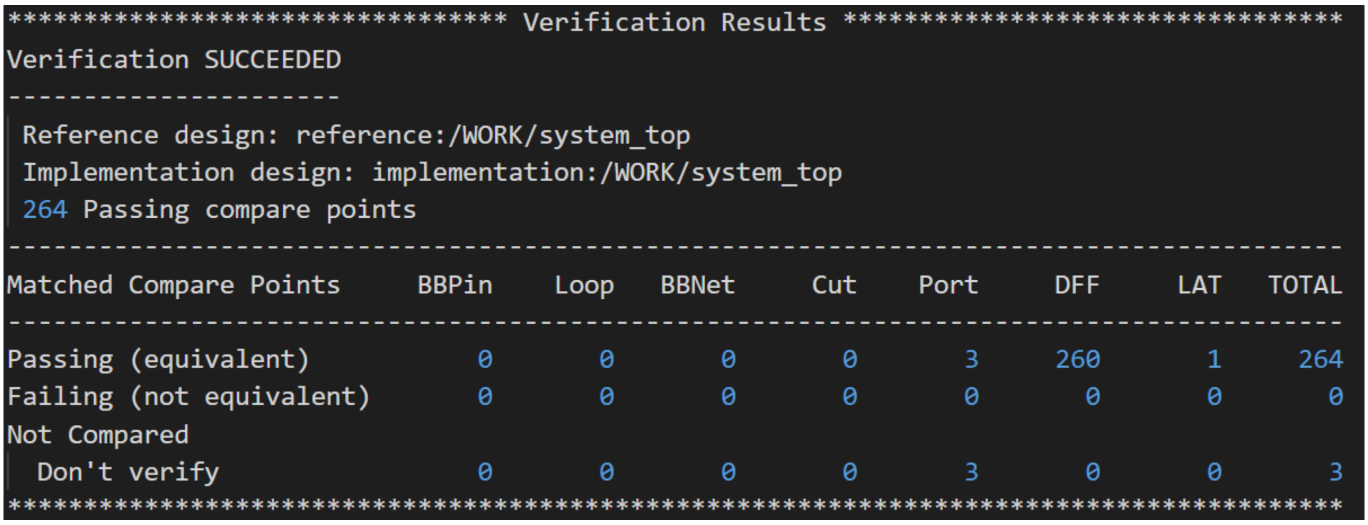
Chip Finishing

Insert filler cells, generate gate level netlist (used in gate-level simulation (GLS)), gate level netlist with PG pins (used for analog IR drop simulation), SDF file (used in GLS), and GDS file which is used in manufacturing of the chip.



Post - Physical Design Formal Verification

Verification Results



```
******************************** Verification Results ********************************
Verification SUCCEEDED
----------------------
 Reference design: reference:/WORK/system_top
 Implementation design: implementation:/WORK/system_top
 264 Passing compare points
--------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
--------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0       3     260       1     264
Failing (not equivalent)       0       0       0       0       0       0       0       0
Not Compared
  Don't verify                 0       0       0       0       3       0       0       3
**************************************************************************************
```

There exist 3 don't compare points which are the 3 output pins of the SO port, they are not verified because they doesn't exeist in the golden RTL files (i.e. their logic was automatically inserted by the DFT tool).

Future Work

Gate-level simulation using:

1. SDF file generated from the physical design process.

2. Testbench used in the RTL functional verification.
3. Gate-level netlist generated from the physical design process.
4. Verilog standard cell library.